Tampere University

Mikko Knuuttila

DESIGN AND IMPLEMENTATION OF WEB-BASED PRODUCT MANAGEMENT TOOL

Faculty of Information Technology
and Communication Sciences
Master of Science Thesis
May 2019

ABSTRACT

Mikko Knuuttila: Design and implementation of web-based product management tool
Master of Science Thesis
Tampere University
Degree Programme in Information Technology, MSc (Tech)
May 2019

Piceasoft is a software company from Tampere, Finland. Main products of the company are different kind of solutions for the whole lifecycle of a mobile phone. Solutions contain services to transfer content from phone to phone, erase data from devices, diagnose device and trade-in. Most of Piceasoft's customers are teleoperators, recyclers, mobile phone retailers and used device vendors.

Piceasoft's software is commercial and is based on licensing. Customers can use their copies of software based on the contract made with Piceasoft. License management needs work from support team and there must be a tool for it. It is common that customers want small changes in used software and this also adds load on support team. Typical needs from customers are related to branding and customization.

Piceasoft's current product management tool, PMT, was implemented in the early years of the company and has collected some technological debt. It can be used only by internal users and for management purposes only. Branding configurations and PDF customizations are made on-demand by developers and require quite much work. Piceasoft needs a web-based solution, which could be used by internal and external users and where customers could manage their products and branding configurations by themselves.

This thesis contains design and implementation of a web-based product management tool. Web application is multi-paged but uses JavaScript library React in pages where interactivity is needed. System has a multitier architecture, where data layer is the new License API. Designed web application does not have straight connection to database but it uses services from License API to get and manage data. Application and the user interface are designed with MVC architecture pattern.

Designed application allows five kind of users to manage and view resources related to licensing. Customers can make branding configurations by themselves without need to make support request for it. Implemented user interface combines set of new features and user flow from the current product management tool.

Keywords: Product management, Node.js, Express.js, web application

TIIVISTELMÄ

Mikko Knuuttila: Web-pohjaisen tuotehallintatyökalun suunnittelu ja toteutus
Diplomityö
Tampereen Yliopisto
Tietotekniikan DI-tutkinto-ohjelma
Toukokuu 2019

Piceasoft on ohjelmistoalan yritys Tampereelta. Sen päätuotteina ovat erilaiset ratkaisut matkapuhelinten koko elinkaaren ajalle. Nämä ratkaisut sisältävät palvelut puhelinten väliseen tiedonsiirtoon, laitteiden tyhjennykseen, laitteiden diagnosointiin ja kauppaamiseen. Piceasoftin asiakkaat koostuvat pääosin teleoperaattoreista, puhelinten kierrättäjistä, jälleenmyyjistä sekä käytettyjen puhelinten välittäjistä.

Yrityksen ohjelmistot ovat kaupallisia ja perustuvat lisensointiin. Asiakkaat voivat käyttää omaa kopiotaan ohjelmistosta sopimuksen mukaisesti. Lisenssinhallinta vaatii työtä yrityksen tukiosastolta ja siihen tarvitaan erillinen työkalu. On myös yleistä, että asiakkaat saattavat haluta pieniä muutoksia käytettyyn ohjelmistoon ja tämä lisää tuen kuormaa. Tyypilliset asiakkaiden tarpeet liittyvät brändäykseen ja kustomointiin.

Piceasoftin nykyinen tuotehallintatyökalu, PMT, tehtiin yrityksen alkuvuosina ja on ehtinyt kerätä hieman teknologiavelkaa. Työkalua pystyy käyttämään vain sisäiset käyttäjät ja vain hallintatarkoituksiin. Brändäys ja PDF-raporttien kustomointi on toteutettu pyynnöstä ja vaatii työtä kehittäjiltä. Piceasoft tarvitsee web-pohjaisen ratkaisun, jota voisivat käyttää sekä yrityksen sisäiset että ulkopuoliset käyttäjät. Järjestelmässä asiakkaat voisivat itse tarkastella omia tuotteitaan ja tehdä brändäysasetuksia itse.

Tämä työ esittelee web-pohjaisen tuotehallintatyökalun suunnittelun ja toteutuksen. Websovellus on monisivuinen mutta se käyttää JavaScript-kirjasto Reactia niillä sivuilla, missä tarvitaan vuorovaikutusta. Järjestelmä on suunniteltu monikerrosarkkitehtuurilla, missä tietokerroksena toimii uusi lisenssi-API. Suunniteltu websovellus ei siis ole suorassa yhteydessä tietokantaan, vaan käyttää lisenssi-APIA tiedon hakemiseen ja hallinnointiin. Sovellus ja sen käyttöliittymä ovat suunniteltu MVC arkkitehtuurimallin mukaisesti.

Suunniteltu ratkaisu tukee viittä eri käyttäjäryhmää. Työkalulla käyttäjät voivat selata ja hallita lisensseihin liittyviä resursseja. Asiakkaiden käyttäjät voivat tehdä brändäysasetuksia itse ilman että tarvitsee ottaa yhteyttä tukeen. Käyttöliittymä on pyritty toteuttamaan siten, että se yhdistää uuden toiminnallisuuden sekä vanhan työkalun peruslinjaukset keskenään.

Avainsanat: Tuotehallinta, Node.js, Express.js, web-sovellus

PREFACE

This thesis consisted of design and implementation of a product management tool for Piceasoft. Thesis was started in in September 2018 and was finished in May 2019.

I would like to thank my supervisors Assistant Professor Davide Taibi and Associate Professor Kari Systä for guiding me with my thesis and giving instructions. I would also like to thank my superiors Samuli Kivinen and Jani Väänänen from Piceasoft for helping me with my thesis.

Tampere, 12 May 2019

Mikko Knuuttila

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

SPA	Single-page application, modern web application design model
MPA	Multi-page application, traditional web application design model
AJAX	Asynchronous JavaScript and XML
URL	Uniform Resource Locator
API	Application Programming Interface, set of defined method for communication between components
UI	User Interface
JSON	JavaScript Object Notation, common language-independent data format
PMT	Product Management Tool, name of the current product management tool
PMC	Product Management Console, name of the product management tool designed in this thesis

1. INTRODUCTION

Piceasoft Ltd is a software company from Tampere. It was established in 2012 and has currently about 40 employees. Company's clients are usually operators, mobile phone retailers, mobile phone recyclers and used device traders and are located from all over the world. Piceasoft offers software for mobile phone's whole lifetime, and consist from three main solutions: PiceaServices PC App, PiceaServices OnTheFly and PiceaServices Online.

Commercial software usually needs licensing system and support team to manage it. There may be many different scenarios where licenses need to be modified or disabled for certain customers and products should be configurated with a centralized system. Web development, networking and application frameworks are evolving all the time and an Internet connection is nowadays very common. These factors make it possible to make software dependent on Internet connection. Many configurations can be managed remotely and retrieved from web servers.

In the early days of web development, there was basically only a one way to implement web applications, the traditional way, where simple HTML pages were returned for every request made from a client. Nowadays, there are more possibilities and especially an application type called "Single-Page Application" has become more popular by making web applications closer to native desktop applications. [1, p. 20] However, SPA approached web application is not suitable for every project and sometimes solution could be somewhere between these two design models.

The objective of this thesis is to design a web application to manage products and licenses of the company. System is designed to support different kind of users that can vary from administrators to customers' representatives. Thesis aims to select a suitable web architecture and a design model that would work with the existing licensing data and interfaces, supports addition of new features and fits to requirements and constraints set by the company. This thesis also consists of design of the user interface that would help the support team to make their daily tasks more effectively and allows customers to configure their products fluently.

The designed tool is part of a company's licensing system renewal, where current licensing system is replaced with a new management tool and license API. This thesis consists

only of design and implementation of the new management tool and does not define data modelling or data migration process. Responsibility of the author was to design and mainly implement the management tool. The project was started in September 2018 and in the beginning of the project, licensing renewal team consisted of two members. Third member joined the team in February 2019 to help in implementation of the designed product management tool.

Chapter 2 consists of backgrounds and requirements of this thesis. Both functional and non-functional requirements are introduced with the current processes related to topic of this thesis. Theorical part that helps to understand this thesis is in chapter 3.

Current product management tool and the problems of it are defined in chapter 4. Design of the new web application is written in chapter 5. The evaluation of the designed and implemented solution is in chapter 6. In the evaluation of the web application, usability of the system and comparison to old system is made with observation containing performance measurements. The final chapter contains conclusions of this thesis.

2. BACKGROUND AND REQUIREMENTS

2.1 Current product and license management

In this chapter, licensing system of the company is described to help understanding how license management works in this environment. Current processes by support team that are connected to this topic are also defined to help understand the problems. Both functional and non-functional requirements for the needed solution are introduced. Finally, all the constraint set by the company for the system are listed.

2.1.1 Licensing system

Piceasoft offers commercial software for companies that consist mostly of teleoperators, retailers, trade-in vendors and recycle companies. The licensing system has many different layers on top of one activated software. Figure 1 describes different layers of the licensing system.

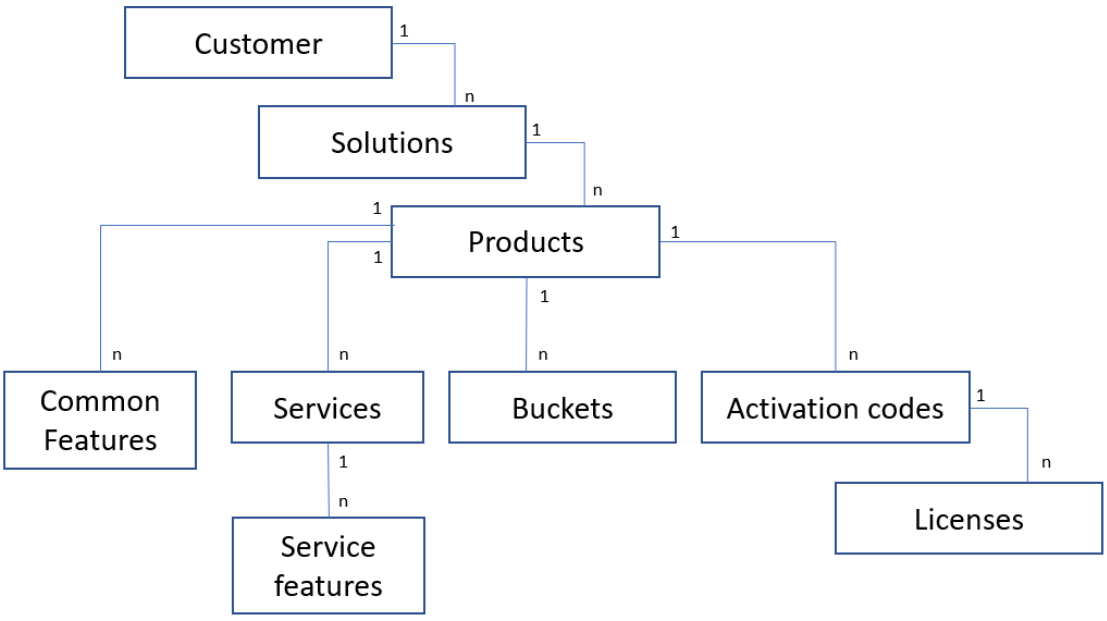

Figure 1. *Structure of licenses for a customer*

The licensing data structure in figure 1 presents the hierarchy of data as it will be in the new system. Old system did not have solution level and previously one product had a list of equal features. New licensing structure divides those features to services, common features and service features like figure 1 presents.

As figure 1 shows, one customer can have multiple solutions enabled. There are currently three main solutions and created products are connected to these solutions. Solution defines what kind of features can be enabled for product. Solution information can also be used to define that what kind of branding can be made for certain product.

Products can have common features set, which will define how the software works. Products can also have different services enabled. For example, in PiceaServices PC Application customer can have a service called “Switch” enabled, which would allow client to open that specific application to usage. Products under different solutions have different services to be enabled. Services can also have service specific features defined, as figure 1 shows.

All products can have activation codes. Activation codes are created for customer and are used in activating the software. Every activation with activation code creates a new license for the client who activated the product. Activation is made on the first startup of the software or when the old license expires. One activation code has a limited number of available activations and can also have an expiration date.

Licenses are created automatically when clients activate products for the first time. However, the expiration date and some other details come from the activation code and only part of the information is from client when he enters the data to the application after giving the activation code.

Buckets are made for customers, who have only specified amount of operations allowed. For example, there could be a deal of 100 operations per week in the contract, and buckets are made to control this kind of logic.

2.1.2 Branding and customization options

Software branding means that customer can define different appearance settings for the application. For example, customer could include their company’s logo in the background of the application or define software’s colours to match their brand.

In Piceasoft, there are two ways of branding the PC application, basic branding and full branding. In basic branding, customer can define only the logo that is displayed in preset locations in the user interface. Full branding is much more precise. When using that, customer can define multiple different small details in all the possible views. Full branding changes needs much more effort and requires more time for developers to fulfil. For this reason, it is not included in this thesis.

End-users can have reports of their operations in PDF-format. For example, if someone makes a mobile phone data switch using PiceaSwitch in some retailer's shop, the software sends a PDF report to email.

It is possible to customize the structure of the PDF-report to wanted format. Usually customization includes showing or hiding some attributes, adding branding logos in the report or even change width or height of elements. Currently, PDF customizations are made on-demand and require development work for every individual case. In this thesis, PDF customization focuses on the visibility of different fields, customer logos and data groups instead of styling the PDF.

2.1.3 Support processes

Current tool for handling licenses is a web application called PMT, Product Management Tool. It was made in the early years of Piceasoft and is mostly used by internal sales personnel, support team members, testing team and some developers.

The main features of PMT are customer management, product management, activation code generation, license handling and search options. Basically, all the operations that are related to licensing are done using this tool.

Branding configuration for customer is currently made on-demand when customer makes a support request for it. Full branding is made less often and is implemented by an UI developer. Basic branding is more common, and the application logo is currently hosted by delivery servers. Branding configuration is made by enabling "product customization" -feature in PMT for each product that customer has and adding customer logo URL or full branding resource file URL in product's feature settings.

The most common case by support team member using the old tool is to search for certain client by having only some information available. For example, only the name of the customer and location of the store is available, and it takes sometimes quite much time to find the correct activated client. After the client is found, usually the next step is to find out what is the used product and what features are enabled for it.

2.2 Functional requirements

Functional requirements describe what the system does or what user could do with it. Functional requirements are typically definitions of system features of behaviours. They can also specify what kind of actions system should do in the background or what kind of interfaces system should be connected to. [2]

2.2.1 Users

Designed system should support multiple different user groups and the system design should support adding new groups in the future. Initially, there would be five kind of users. Admins and support team members are always internal users and they have almost the same access rights. Salesmen are usually internal but there is a possibility of being external members. Resellers and customer representatives (normal users) are always externals.

Customer's representatives can manage only products that are owned by the customer. Salesman and resellers can manage customers and their products that belong to their area of responsibility. Support team can manage all customers and their products. Admin users have all access rights. The access control logic should be maintainable and permission granting to specific groups should be possible with easy steps.

2.2.2 Customer handling

All the customers should be listed for user with all information easily available, if user has access for customer data. If user has access only for one customer's information, user can't access list of all customers. User can edit customer's basic information.

All the enabled solutions for currently selected customer should be shown in the solutions -page. If the customer doesn't have any specific solution enabled, User with adequate access rights can enable solutions for the customer. In this case, a new product is added for the customer.

Admin users should have easy search actions to find wanted customer by using different kind of key values. Often these key values are different id-values, like product-id, client-id or activation code or name values like customer's name, activation code's name, license's name or store name.

2.2.3 Product, activation code, bucket and license handling

All the products for the customers are listed and can be viewed. All the features for the product are also shown. User with adequate access rights can add, update and remove products for the customer and is able to manage enabled features and services for products.

User should be able to list all customer's activation codes by one product. User with adequate access rights should be able to update attributes, delete existing codes and generate new activation codes with wanted attributes.

It should be possible to disable, enable and edit licenses for admin and support users. Customers can list their own licenses and see attributes related to it. Licenses should indicate the expiry status clearly.

Admin and support users should be able to make mass operations to products or activation codes. They could edit attributes of many products at the same time. They should also be able to export customer and product data from the system.

2.2.4 Branding configuration

User can define customer's branding logo for all the products owned by the customer or only for specific products. Branding editor should show a preview of the application with added logo. All the solutions should have their own previews that match application's real appearance. User can also delete or modify existing branding configurations. Only basic branding is possible to be added. Branding editor should support more branding options in the future without need to make major changes.

2.2.5 PDF report customization

User can configurate PDF reports for all customer's products or just for wanted ones. User can change existing configurations or delete wanted customization configurations.

PDF customization should be made in the same branding editor where the customer logo branding is made. User should be able to see a preview or sample PDF with the wanted customization settings without saving them. User should be able to edit report fields for different report types and preview PDF reports of all report types.

2.2.6 Feature configuration

User can list enabled services and features of products. Administrates can easily enable or disable services and features from customers and add application settings for enabled features.

2.2.7 API keys management

Web application should list all the API keys owned by customer. Keys should be grouped by application type where the key is for. Customer can view their own keys and other attributes related to the key. API key value should always be hidden by default and viewable only by clicking "show key"-button. It should also be possible for admin and support

team users to see client keys and other attributes from all the customers. Only admin users and support team members can generate API keys for customers.

2.2.8 User guide

Tool should have a good user guide page that covers all the main features for all user groups. User guide should be accessible from the website and tooltips should be available for offered actions. User guide texts should be editable easily by maintainers using a text editor.

2.2.9 Use cases

Use case diagrams can be created based on the functional requirements. This section contains use cases for five user groups. Figure 2 shows use cases for a basic user

.

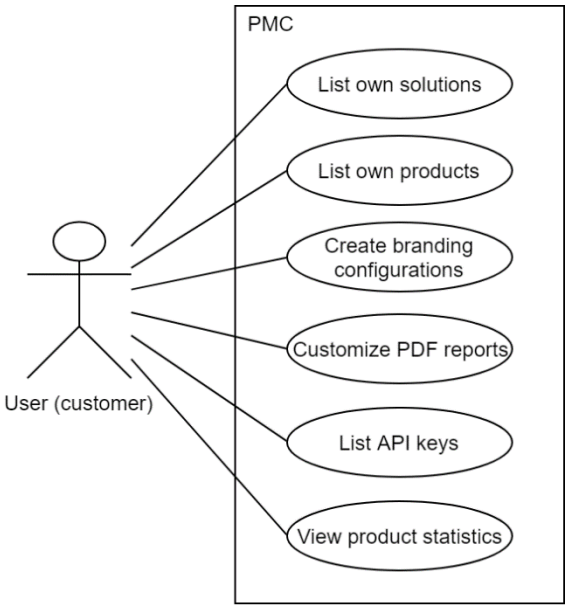

Figure 2. *Use case diagram for a basic user.*

As figure 2 shows, basic users can list their own resources and create branding configurations for their own products. Product statistics contain information from customer's own products. Figure 3 describes use cases for support team members and admins.

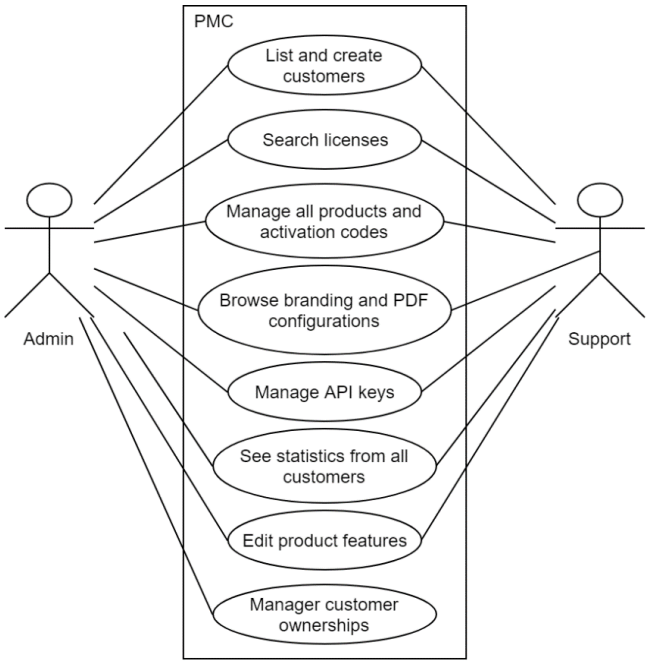

Figure 3. *Use case diagram for admin and support users.*

Admin and support team members can manage all the resources, like figure 3 shows. Only thing that support can't do, is adding customer ownerships to sales and resellers. Figure 4 introduces use cases for sales and resellers.

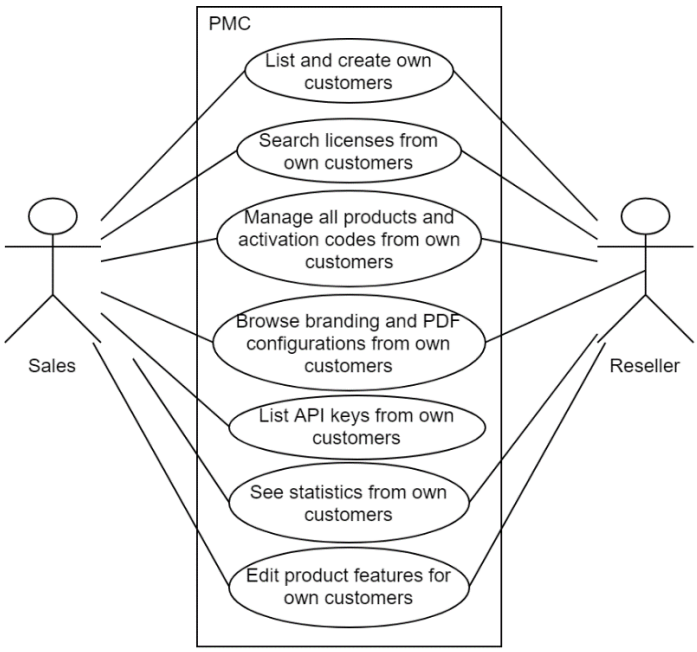

Figure 4. *Use cases for sales and resellers.*

As figure 4 shows, sales and resellers are mostly the same kind of users, but resellers don't have access to their customer's API keys. Sales users can list API keys for their customers.

2.3 Non-functional requirements

Non-functional requirements define more technical aspects of the IT system, which have an influence on the reliability and performance side instead of the features and business logic of the software. Compared to functional requirements, non-functional requirements define that how the system should work instead of what the user could do with the system [2].

2.3.1 Performance

Average system response (ART) times should be under 500ms for all users from any locations. Peak response times (PRT) should be under 3000ms for all users from any locations. System should be able to handle multiple concurrent users without any notable effects for users and this should be considered when choosing system's architecture and technology choices.

2.3.2 Security

System should use commonly acknowledged best practises in web application security issues. System should be designed to avoid all the security threads mentioned in the latest OWASP's top 10 application security list, which can be considered as a trusted source of security issues. System should handle customer information carefully without leaking any sensitive information outside from the application. Access control should be easily managed and suitable for this environment. Vulnerabilities in 3rd party libraries or packages should be detected as soon as possible.

2.3.3 Stability

System should be error tolerate and should be able to restart itself if error occurs. This should be considered in deployment design.

2.3.4 Availability

System should be available for 99,999 % of the time.

2.3.5 Maintainability

System should be maintained without downtimes. This means that library updates and configuration changes that need application restarts or pauses could be executed without notable downtimes.

2.4 Constraints for the system

There are many things that effect on what kind of architecture or approach is reasonable to be chosen in software project: project schedule, individual skills of team members, currently used technologies and future vision of what kind of technologies is wanted to be used. These all can be considered as constraints.

In software development context, constraints are boundaries and restrictions for the developed system. Constraints are usually set globally, by company standards or by management. Reasons behind constraints can be technological, financial or environmental. Politics or legislation can influence on constraints too. [3]

In this thesis, there are few constraints that influence on the design of the system and must be considered when selecting architecture of technology options. Constraints are shown in table 1.

Table 1. *List of constraints for the system*

Type	Explanation
Database	Even though the system has no straight connection to database, usage of relational SQL database in License API eliminates benefits from some architectural options the benefit of having document database like MongoDB in the back-end.
Data modelling	All the data is mainly handled by the License API and contains a lot of legacy data structures. Data migration plan is part of development of License API and is not included in this thesis.
Server-side technology	Node.js is currently the main server technology in the environment and the system should be designed to be executed in it.
Project schedule	The system should be up and running in targeted timetable, which effects on chosen technologies and application types. If the web application structure or technology is completely new, the implementation and testing of the system could take considerably more time.
Legacy tool	The current product management tool, PMT, has some features and user's workflows that should be considered when designing the user interface of the new management tool.
Deployment	Deployment host provider should be the same as for the other web systems.

Table 1 introduced the main constraints for the system. Architectures of the current internal systems must also be considered so that the system is as maintainable as possible for developer team and IT department.

3. THEORY

This chapter aims to explain the basic principles of web development and terms related to it. The most common technologies and architectural options are also introduced to help understand the choices made in the design of the system.

3.1 Web development

Web development usually refers making websites that are available on Internet and are hosted by web servers. Users can access websites with web browsers that can show the data sent by a web server. Web development usually consists of developing logic to client-side, server-side and database but coding even a simple text file with working HTML code can be considered as web development. [4] Many popular web applications like e-commerce sites or social media platforms consists of big and complex server-side applications with huge amount of data and a lot of logic also in client-side.

3.1.1 Client-server architecture

Client-server architecture model is a networking structure where multiple clients use services from a centralized host server. Client sends a request to the server, which handles it and responses with data or just status code. Usually clients and server are in different networks, but it is also possible that they are in the same environment. To achieve successful communication, client and server must use the same protocol and rules in messaging. [5] Figure 5 presents client-server architecture.

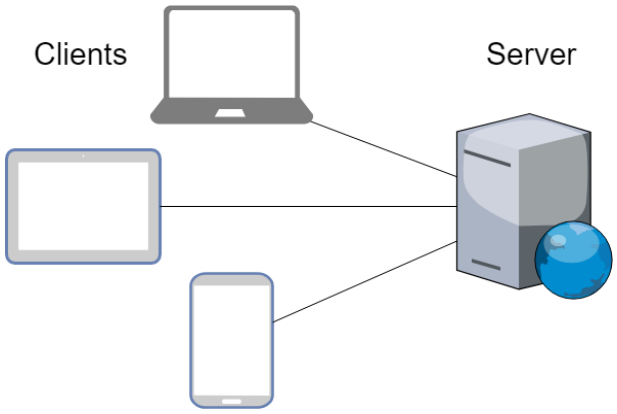

Figure 5. *Client-server architecture described.*

As figure 5 shows, many kinds of clients can be connected to server and usually one server can serve multiple clients at the same time. Server can also reject the connection

made by the client. When accepted, client is using the same protocol and connection is made. Even though one server can handle multiple concurrent connections, a traffic limit still exists. Therefore, many web services are distributed to many different servers to increase the number of handled clients. [6] Web services can also be distributed to many servers to ensure high availability.

3.1.2 Web application design models

When designing a web application, there are few different approaches available. The traditional way is to use server-side rendering with multiple pages. In this approach, client's browser makes a request to server and HTML page is rendered and returned using wanted data. Making a request and receiving HTML file in response makes a page refresh on client's browser and with a lot of content, it can take some time. This approach is also called as multi-page application, MPA. [7]

The opposite pattern is to make a single-page application, which is a very popular nowadays when making a website with interactive user interfaces. Single-page applications, also known as SPAs, render content dynamically to the user. Single-page apps need only single page load and most of the application logic is implemented with a JavaScript using some of the many available front-end frameworks. The initial load of all JavaScript, HTML and CSS files is heavier but after that, no unnecessary HTML rendering is done from the server-side code. [7]

However, selecting web application design model is not that simple. There are many kinds of solutions between MPA and SPA. Even though usage of SPAs has risen a lot in the past few years, not all applications are made and even need to be single-page applications [8]. One hybrid approach is to use server-side rendering in some parts of the website and include JavaScript applications to parts where application needs to be dynamic and interactive.

Hybrid applications could also use some libraries, like *Turbolinks*, to prevent page reloads when moving from one page to another and replace only the content of the HTML's body. By this way, user could get a faster and more interactive experience and web application could still use server-side rendering [9].

3.1.3 Web architecture patterns

Web applications can be designed with different architectural patterns. Design model selection can influence the selection of server-side architecture too. One way to make

web applications with MPA approach is to implement a monolithic server-side application. Monolithic applications contain all the needed business logic inside same application and offers APIs to those who needed and at the same time offers actions for web user interface [10].

Multilayer architecture is a pattern where different parts of software are divided to layers. All layers provide specific kind of functionality. In web context, application could be divided to three or four layers: presentation layer, application/business layer and data layer. Presentation layer is responsible for showing results to user, application layer handles inputs and routes them correctly, business layer manages business logic and data layer handles queries from data storages. Multilayer architecture is usually used with client-server architecture. [11]

Microservices is a pattern to make applications with small logical units. It divides application to smaller logical parts which all provide interfaces to manage and read specific data related to that service. It is a good pattern if designed application contains different kind of information that are not related too tightly to each other. Microservices help deploying small parts of application at a time and enables development of different areas separately. Microservices is a good choice as a back-end architecture for a mobile application or single-paged web application. Microservices usually have partitioned databases which can be a problem in a complex database. [10]

Serverless is an application execution model where some parts of the application, usually functions are dynamically executed in the cloud platform. It is also known as *Function as a Service*. Using serverless-strategy can be a cheaper choice compared to normal web application deployment because application is executed only on demand and uses resources dynamically. Serverless code is started again every time it is executed, so it is stateless. Serverless approach requires that application is based on functions. [12]

3.1.4 Model-View-Controller architecture in web applications

Model-View-Controller is an architecture pattern to design software with user interfaces. It consists of three different layers that have different roles. Model-layer is responsible of data structure, data handling and business logic. View-layer shows the data for user and offers actions. Controller-layer handles the user inputs, passes data to models and updates views. [13]

In web applications, model is usually related to database handling and business logic around it. Routing handles passing the request to the right controller function in the web application. When data is queried from database and processed to wanted format, it is

passed back to controller, which passes it to views. Views render and return the HTML pages to the client. MVC is commonly used architecture pattern in this company and all the web developers in the team are familiar with it. MVC can be used with single-page application or multi-page application approach. Design model selection defines where controller and view logic are managed.

3.1.5 Common web technologies

When talking about web technologies, JavaScript, HTML and CSS are the most important ones. These three are the core technologies when making websites and web applications.

JavaScript is the programming language that can be used in web pages and works in all browsers. It was created by Netscape in 1996. Developers can add dynamic functionality to web pages with scripts written in JavaScript. Scripts are programs that don't need to be compiled and are executed in browser when the website is downloaded. [14]

HTML is a markup language for websites and web applications. It is used to show content in a web page to user. Elements in HTML are defined with tags. Tags are not shown to user but are used by the browser when interpreting the page. [15]

CSS is a style sheet language for defining how elements are shown in a document written with a markup language. CSS helps separating formatting and actual content and makes possible to make different rules for different kind of devices. [14]

3.2 Networking

Hypertext Transfer Protocol, HTTP, is a protocol that is used to transfer website data between clients and web servers. When client sends a HTTP request to web server hosting the website, it responds with a status code and wanted data [16]. HTTP requests support different kind of methods, like GET, POST, PUT and DELETE. These define that what kind of operation is wanted to be made. For example, GET request gets a resource like HTML file from web server whereas POST request creates a new resource and PUT request updates an existing resource.

Secure version of HTTP is HTTPS, which uses secure socket layer to make the request secure. Usually any website where sensitive data like login information is passed, uses HTTPS. If website uses only HTTP connection, it is possible that someone could hijack information inserted on the website. Websites using secure connections can be checked

by looking at the URL of the site. If there is “https://” before the actual web page address, it uses secure connections. [17]

3.3 Web application deployment

Once the development of the web application or website is finished, the next step is to make application publicly available for usage. For this, web application needs an environment which hosts the application. [18] Deployment can be made for development or production purpose. When deploying to development environment, application in current state is usually still under testing and handles development data instead of real production data. When deploying in production, application is made available for end-users to manage real production data.

Deployment environment needs hardware, operating system, required software like runtime environment and 3rd party libraries and database if used [18]. Hosting a web application is possible with own servers but is much easier to host in some hosting service which offers hardware, software and stabile network connections to ensure that the web application is up and running without need to own active monitoring.

Hosting providers can run offer entire web server infrastructure with management tools to change configurations. This kind of service model is called Infrastructure as a Service (IaaS) and is used when user or company deploy many kinds of applications and wants a full control of the infrastructure. If the selected service model is Platform as a Service (PaaS), hosting service takes care of the application stack like operating system and system updates and offers usage of the platform and runtime environment for hosted web applications. [19, pp. 13-18]

3.3.1 Cloud services

Cloud computing means offering or enabling on-demand network access to use computing resources that can be configured dynamically and are used from a shared pool of resources like storages or applications [20]. Many enterprises host their web applications and services using cloud services.

Most popular cloud services for hosting public services among enterprises are Amazon Web Services, Microsoft Azure and Google Cloud. Other major services are IBM Cloud, VMWare and Oracle. AWS has been the most popular for recent years but services from Microsoft Azure has grown much in the past years. [21] All the services are available for both individual users and enterprises.

The company uses Amazon Web Services and especially Elastic Compute Cloud (EC2) -cloud service for hosting web services. Compute capacity can be changed dynamically to adapt the needs in EC2. It also provides a web interface to manage the resources needed in computing and allows usage of extra server instances just for short periods of time if needed. The payment of the computing is only from the capacity that was eventually used when using on-demand instances. There is also a possibility to use reserved instances, when the usage is known to be steady for hosted applications. [22]

4. CURRENT PRODUCT MANAGEMENT TOOL

Current product management tool was implemented in the early years of the company and has been the main tool for handling customers and licenses. This chapter aims to introduce structure, features, architecture and technological choices of the application. Problems of current system are described so that they can be avoided in the design of the new system.

4.1 Website structure and features

Current product management tool, PMT, focuses mainly on license management. It contains customer listing and their management. Users can list and edit customer's products, activation codes, licenses, buckets and features. UI consists mostly from tables and dialogs. Tool offers data export commands in Excel format and has good table browsing actions like sorting and column visibility toggling.

PMT offers search page to look for specific licenses. It accepts three kind of input types: client id, product id and activation code. Results are listed in a table and can be edited right away. Search view and UI layout is shown in figure 6.

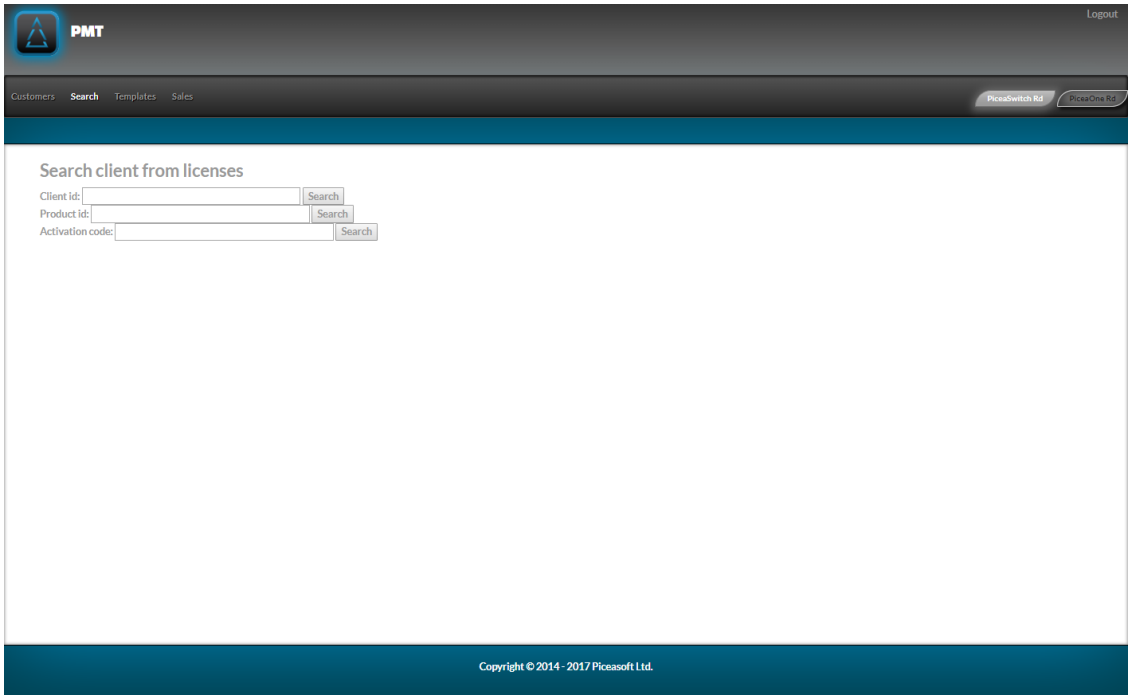

Figure 6. *Search view from PMT*

Figure 6 shows the basic layout of the website. Navbar has links to main actions and user or database related things are in the top right corner. User Interface is very simple, and no extra descriptions or actions are offered. Old tool also lists sales personnel details, but this feature will not be included in the new system.

4.2 Architecture and used technologies

PMT is a multi-page application. Server renders and returns HTML page for every request. Server-side of the application is implemented with PHP and uses a framework called Laravel. Application is designed with MVC architectural pattern. It has a straight connection to licensing database and updates data with SQL queries.

The user interface of the tool doesn't contain very much interactivity. Most of the pages list data and offer simple actions to them. It uses JavaScript library jQuery for some functionality like table handling, dialogs and small UI changes. Tool has two user groups: normal user and admin users. Normal user does not have access to sales personnel listing or all the details in the view. Application is only used by internal users.

PMT does not only offer user interface to manage licensing data. It also has commands for other applications to ask licensing data and make license checks. So, at the same time, PMT offers an API and UI for license management.

4.3 Problems in the system

There are many reasons why new product management tool is needed. The biggest individual factor is that the current tool has gained technological debt. PMT uses an old version of application framework library Laravel, which uses an older version of PHP. Newest version of Debian Linux, which is the used operating system in host environment of PMT, comes with newer PHP version and the used version of Laravel does not support that version of PHP. Updating current tool to use newer version of Laravel would require many days of development work.

The architectural structure of the current licensing system is also challenging. PMT offers both web user interface and application programming interface at the same time. This kind of structure is hard to maintain. Dividing licensing management to two different applications is necessary to retain system maintainable.

PMT has currently only two user groups: basic users and admins. Adding new groups would require too much re-implementation of application logic. Design of new management tool is necessary because a tool where customer’s users could manage their products is needed.

5. DESIGN AND IMPLEMENTATION OF WEB APPLICATION

There are many things that must be considered when designing web applications. All the choices effect on how long the implementation will take and how maintainable the system is. Rossi et al. mention [23, p. 17] following things that must be taken into account in web application design: system usability, performance, responsiveness, security, integrity, maintainability, testability, accessibility, support for localization and page design.

Besides of designing web application, also the architectural selections behind the application must be made. When selecting the architecture of web application, Rossi et al. propose [23, p. 24] functional requirements, non-functional requirements, technical issues and experience as main things that have an effect on the selection.

When the design of the web application was ready, the implementation process started. Implementation was done in smaller parts, which were individual features of the system. During the implementation, demo sessions were held to gather feedback from internal users and other employees. Internal users also started to test the web application when the system contained enough features to work properly.

5.1 Architecture

Web architecture consists of system architecture, application architecture and software architecture. System architecture defines the environment of the whole system, including the network, other servers and the databases that are used. Application architecture is a little more specific definition of the modules of the application. Software architecture describes what software modules are needed in modules made in the application. [23, p. 24] This chapter describes the architecture of the designed web application in system and application level.

5.1.1 System architecture

When selecting architecture of the system, all the other services and dependencies must be considered as well. A new licensing service is implemented at the same time and data handling is managed by using its API. For this reason, the main architectural choice for the system is a multitier architecture. System will have a presentation layer, application and business layer and data layer. Data layer is the licensing API and this thesis does

not contain design of that service. Data layer is physically separated from the application layer and this suits well for the principles of multilayer architecture.

Microservices and serverless strategy would have been good alternatives if design of data management would have been included in this thesis. But because designed web application uses predefined interfaces for data management, architectural selection focuses more on application logic and data presentation.

Designed web application will be deployed in a cloud network managed by the company and all the used internal services by the system will be in the same network. Application also uses Login-server for many operations, like authentication and API key management. System's clients and server environment are described in figure 7.

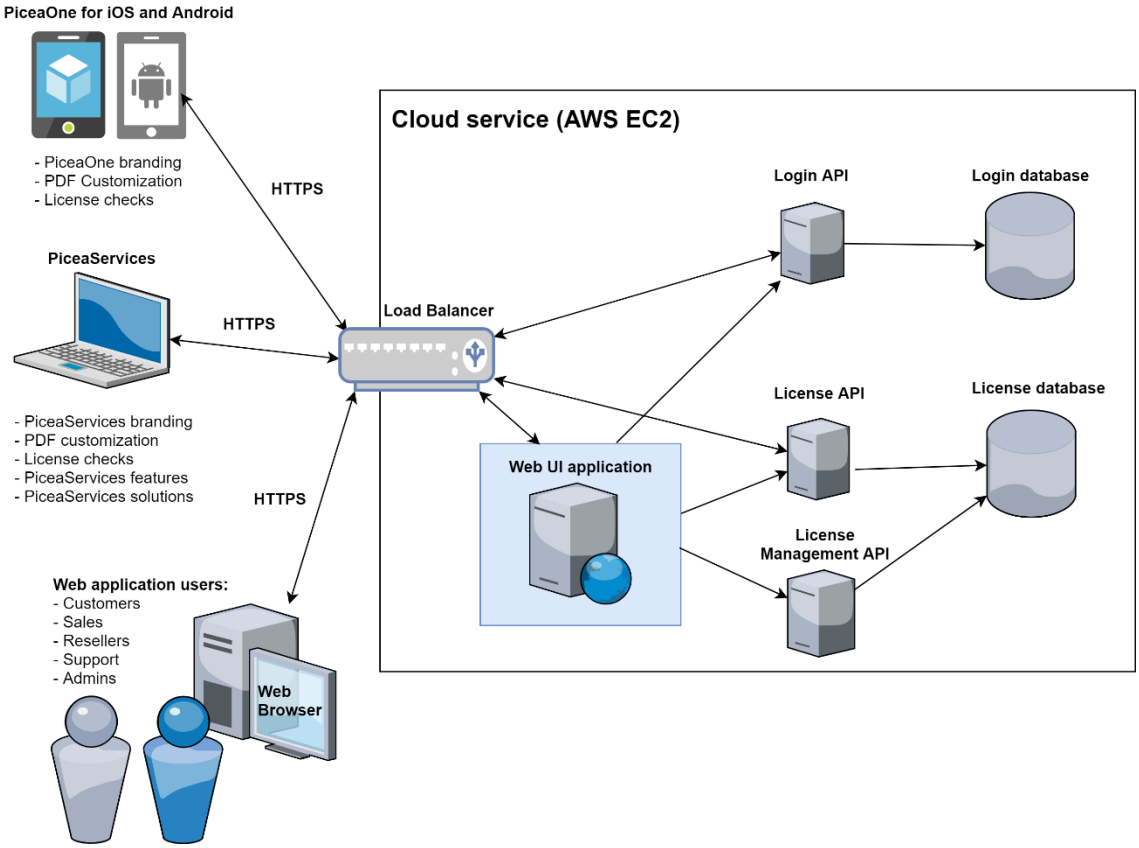

Figure 7. *Infrastructure overview*

As figure 7 shows, designed web application and License API have three kind of clients. PiceaOne mobile application uses License API to retrieve PiceaOne branding data, get PDF customization data and make license checks. PiceaServices PC Application uses License API to get branding data, PDF customization and features. It also makes license checks. Clients of the designed web application itself are users with web browsers.

5.1.2 Application architecture

Multitier architecture was selected as the system architecture model and MVC as a design pattern for user interface design, handling the data and controlling the application logic. Nginx proposes a usage of facades when making a web application with MVC architecture and usage of microservices as models [24]. Even though data layer is not only microservices, this kind of design can be good when having other services responsible for data management. MVC architecture of the web application is shown in figure 8.

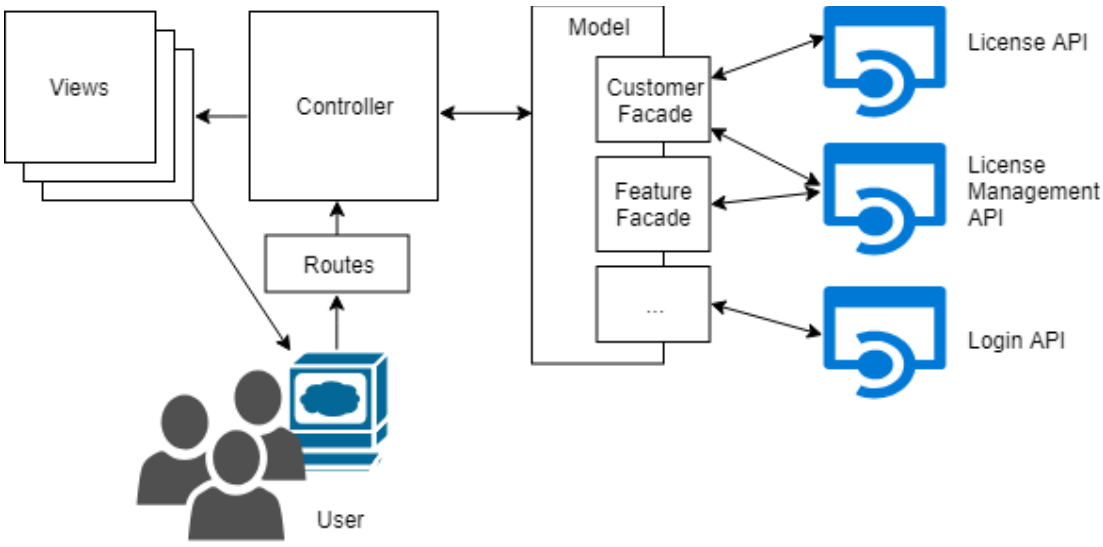

Figure 8. *MVC architecture in the designed web application.*

As figure 8 shows, the main structure of the application is very similar to typical MVC web application. The main difference is in the model section, where is a set of facades instead of straight database connection. Facades are classes that describe available resources and offer interfaces for controller to read, write, update and delete wanted data. All the facades are using API's from other microservices that are responsible for storing the data. Facades also handle error situations that come from the used services and a backup system can be build there.

Figure 8 also shows how request moves from client to routes, controller, model and finally to views. Route handler matches the target URL of the request made by the client and passes it to correct controller function [25]. It then handles the request data, retrieves needed data from model and passes the data to the views, which then renders user interface changes to user's browser. Sequence diagram of typical request is shown in figure 9.

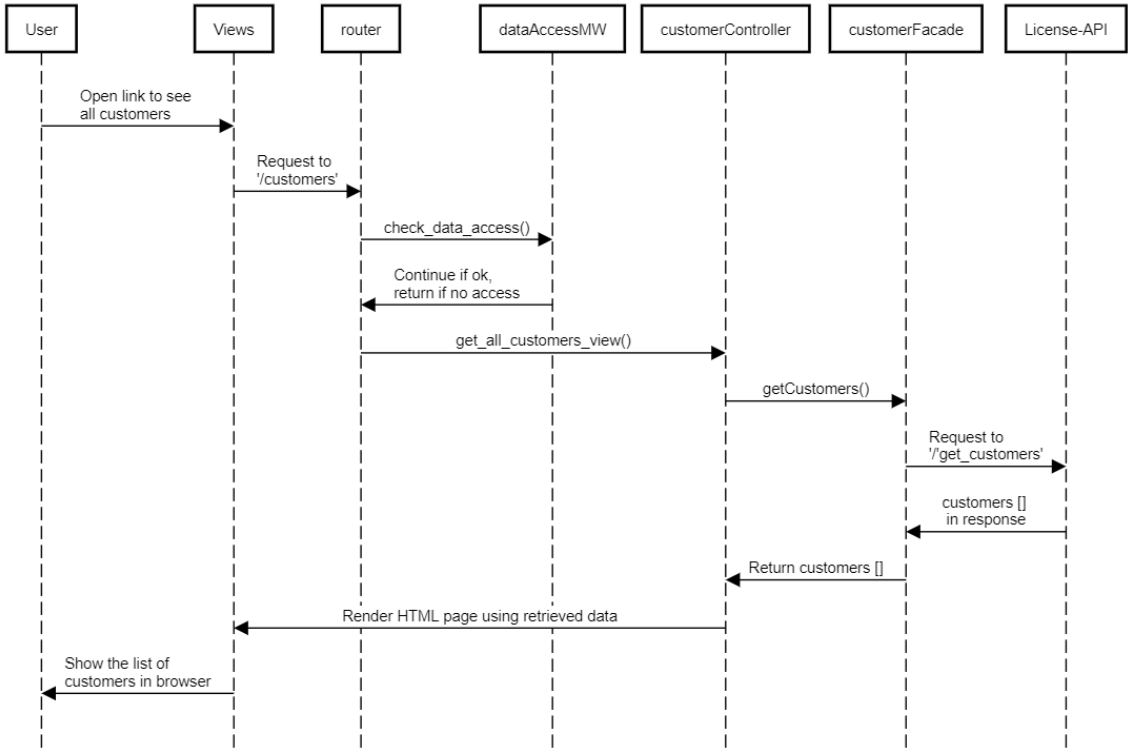

Figure 9. *Sequence diagram for handling request to get list of all customers*

In figure 9, user opens a link from HTML page, which makes a request to the web application. Router gets the request and first makes a data access check to validate that client can access this route or resource. After all the needed checks, it passes the request to *customerController*, which uses *customerFacade* to get a list of all customers. *CustomerFacade* uses Licence-API to get the data and returns it if the response is successful. *CustomerController* passes the customer list to views for HTML page rendering. Finally, the rendered view is shown to user.

Both controller logic and views can be in client-side or server-side in this application. Interactive pages made with *React* have views and some business logic in client-side, because HTML is rendered dynamically to browser without page reloads. Still, most of the controller and view logic is in server-side because of MPA-oriented design of the application. Because of this, in most of the cases views render and return an entire HTML page to user.

5.2 Web application type

Chapter 3.1.1 introduced the most used web application design types: SPA, MPA and the hybrid versions between them. When selecting between these design models, also the constraints of the system should be considered to make a suitable choice for the design of the solution. Constraints for system are shown in chapter 2.4.

5.2.1 Comparison of application types

When comparing these two types, it is good to have some topics in the help of the comparison. For this kind of management application in scheduled project, there are few factors that need be considered. Website must have clear navigational structure and should support direct links to resources. Effort and time for developing application shouldn't take too much from the current developer team. Application should be secure because of handled sensitive customer data. Performance levels are defined in requirements and should be considered in the comparison. Other benefits of both application types should be also mentioned, because they could have an effect to selection even though they wouldn't relate to any of previously mentioned sections.

Navigation and links

Multi-page application is suitable when a clear website navigation structure is needed [26]. Using MPA, linking to wanted resources or search queries with used options is part of the core functionality of the model.

Routing logic is also supported in most of the popular SPA oriented libraries, like *React*, but usually they need more development and configuration in the client-side. When using client-side routing, every view would have a navigational URL in the browser like in multi-page applications and would show content based on the route without full page reloads. [27] This would also allow usage of links to specific resources in SPA but would not completely remove the need of server-side routing, because the data requests made by the client-side application would still need routing in the server-side.

Development

Developing single-page application needs ability to handle much bigger entity because of the nature of being closer to native application. For this reason, it takes more time in total to be developed [1, p. 9]. Especially for teams that don't have experience on SPA-oriented libraries or frameworks, design and implementation of a single-page application can be much bigger project than making it as multi-page application.

When developing a multi-page application using MVC pattern, adding new routes is simple and doesn't usually have any side effects on other routes because the controller function is separate from others. In single-page applications, adding new routes increases the size of client-side application. [28] This makes adding new members to development team easier and there is no need to understand the full complexity of the client-side application.

Security

Authentication and authorization are made in server-side in both approaches [1, p. 8]. JavaScript is not a compiled language. In SPA, most of the business logic and data handling is made on client-side, which makes cross-site scripting easier for attackers [29]. Multi-page applications have usually data handling in client-side too and are not immune for it, but the amount of sensitive business logic in client-side is usually much smaller.

Application with many different user levels needs to check access rights for all the operations. Access rights can be made in server-side in both models when retrieving data. But if attackers found vulnerabilities in client-side logic, there is a possibility that some actions or sensitive data can be found. Hiding the business logic is more difficult if it is made in client-side. In MPA approach the business logic and data handling is made in server-side and is controlled more easily.

Performance and user experience

Single-page applications load all the JavaScript and CSS in the initial load of the web application. When the initial load is done, no unnecessary page reloading or fetching is made. [7] Only those parts that have changed are rendered again. This makes application faster and more user-friendly. Almost all the requests to back-end are made to retrieve some data from back-end or to make some changes to existing data. This is faster than moving HTML files all the time [30].

However, some SPA-oriented solutions can be quite heavy if there is a lot business logic with JavaScript on the client-side [30]. Applications with a lot of small components that load data from server-side at the same time may be slow in the initialization of the web application because of the continuous processing by the browser.

Multi-page applications load only necessary JavaScript files that are defined in the received HTML file. If server-side rendering of some part of the page takes a bit longer, user needs to wait for rendering to finish and HTML page to be sent. This has an influence on user experience. With smaller amount of data, HTML files are rendered relatively quickly and navigation in the web application can be fast.

With some libraries, like *Turbolinks*, server-side rendered HTML file is fetched in client-side and merged to existing webpage without any page reloads [9]. This kind of hybrid approach would prevent page reloads and would allow business logic and rendering to be in server-side.

Single-page approach offers possibilities to easily make an interactive user interface. Because most of the business logic and state management is handled in the client-side, no state savings via page reloads are necessary and editors or other features can be implemented to be used like native applications. However, it is also possible to make an

interactive user interface in multi-page applications too. It is possible to add any size client-side JavaScript applications implemented with *React* to a website [8]. Web application can be multi-paged but also have some pages of the website interactive by adding logic from *React* or other same kind of UI -libraries to achieve better user experience. For example, dashboards and editors could be implemented with SPA-oriented technologies.

Maintainability and testing

Testing is an important part of software development. In both models, back-end testing can be made with same techniques. System testing is made using web user interface in both cases and the type doesn't have an influence on it.

In single-page applications, testing back-end is mostly API-testing where all the back-end routes retrieve and response JSON data and is easily verified. However, if most of the business logic is made in client-side, testing it automatically needs more configuration and might require usage of front-end testing framework.

When testing back-end of a multi-page application, it is easy to verify that server responds to all routes in the application. However, if there is no specific API-methods available for getting data, it might be difficult to verify that the retrieved data in GET-requests is correct. In other types of requests, like POST, PUT and DELETE, automatic API testing is much easier, because responses are often sent as JSON. Testing back-end automatically helps verifying that business logic work if it is made in server-side.

Other mentions

Search engine optimization is often mentioned when comparing MPA and SPA. When rendering in server-side, it's always mentioned as a strength of this application type. For SPA-oriented applications it is also possible but much harder and needs extra effort to make it work as wanted. [26,28,30]

Single-page application usually have an API in the back-end, where all the needed data is retrieved from. If a mobile application with same client-side functionality is needed, the same back-end API can be used. [30,31] This is not possible in MPA where server renders and returns HTML and developing a mobile application would require a new back-end.

5.2.2 Selecting application type

Chosen approach to design the web application is multi-page application with usage of front-end JavaScript libraries and frameworks for making user interface user-friendly and

interactive in the parts where it is needed. As React documentation says, not all websites need to be designed to be single-page [8]. Designed management tool has features that require interactive UI only in some parts of the application. It is possible to implement an interactive UI-application to only these pages using SPA oriented libraries and frameworks. Most of the data shown in the web application is listing resources in tables. Resources that can be modified or deleted require bit of logic and AJAX calls in the client-side but can be made with lightweight JavaScript libraries.

Gitlab chose [32] similar approach in their front-end plan. Their different pages have own applications made with Vue.js. In this approach, only the needed JavaScript files are loaded in each page and the front-end application logic is divided to separate applications. Those parts that are lightweight and would not need any JavaScript, wouldn't load all the JavaScript code in the web application.

Designed solution has multi-level page structure. With MPA approach, the website will have a clear site navigation structure which is maintainable and easily developed. Linking to specific resources and searches is important and it is core features of MPA approached web applications, even though single-page application libraries can handle routing too.

In the future, the system will have new pages and routes. In MPA approach, adding new pages doesn't influence performance or make application logic more complicated. The nature of the application needs to have a good security side. It is necessary to be very careful with the visibility of the data and the access control. It is safer to do that in server-side. It would be a risk to manage action permissions in the client-side because application has multiple different user groups with different kind of access rights.

Neither search engine optimization or back-end reusability were not important when selecting the application type. Application will be used by specific group, so it doesn't have to be found from search engines. There won't be any mobile application in the near future for the same purpose because of the nature of the system.

Many JavaScript libraries used in SPA oriented websites have support for reusable UI components when rendering HTML to user. Usage of this kind of components improves maintainability. Many libraries in server-side, like *Embedded JavaScript templating (EJS),* also support this and can be used in multi-page applications when rendering HTML from templates in server-side.

Development of the web application in schedule and skills of the developer team also affected to selection of the application type. SPA-oriented web development is not a familiar approach to developers of the application and it would have been too big risk to

choose it. However, choosing to implement some parts of the application with *React* helps team to take some steps towards this approach and makes it easier to start making single-page applications in the future.

5.3 Used technologies

5.3.1 Back-end

The main technology in the back-end of the web application is Node.js. Node.js is a runtime environment for JavaScript and is built on top of Google's Chrome V8 JavaScript engine [33]. Web applications for Node.js are written in JavaScript. The approach from Node.js is very different to traditional server-side technologies. It is event-driven and has non-blocking IO operations. Instead of using new OS threads for every request, Node.js has an event loop to handle multiple connections concurrently. [34]

Back-end of the web application could be made without any JavaScript-frameworks, but it is often better to use widely used frameworks to help with commonly used operations like handling requests and making responses. Express.js is one of the most used frameworks for Node.js based web applications. It is quite minimalist and flexible but offers many web application features, HTTP methods and routing that gives a good basis to construct working and fast web application. [35]

Because designed web application is multi-paged and uses mostly server-side rendering, a template engine is needed. Embedded JavaScript Templates, EJS, is a lightweight template engine that uses plain JavaScript and is used for server-side rendering in the application. EJS supports dividing different pieces of template to separate files so that they can be reused.

Session management is mostly handled in server-side when application is multi-paged. If there are multiple instances of executed web application, it is good to have only one in-memory database, which is separate from all the other applications. *Redis* is an in-memory data store which supports wide area of different kind of data structures [36]. When user logs in to web application, session is made and stored to Redis. During user is active, all the session-data is updated to Redis. If application goes down or user is changed to use other instance of it, Redis storage stays and offers the same session data. [37]

5.3.2 Front-end

There are many popular JavaScript libraries and frameworks to use in front-end. Some libraries are mostly just for user interface design some of the frameworks are complete MVC-framework for the front-end logic. However, all of these can be used as much as developer wants.

Bootstrap is a client-side framework for creating responsive design to web application. It includes HTML templates, CSS definitions and JavaScript plugins for many elements. With Bootstrap, it is easy to make all web applications to work well all the devices with different sized screens. Bootstrap 4 is the latest version of the framework and is used in this project. [38]

jQuery is a fast and widely used JavaScript library to manage DOM and do common methods like using AJAX more easily [39]. Bootstrap 4 requires jQuery, so it is a natural choice to include in this project.

Some views in the application will require good UX and logic in front-end. There are many options for achieving these targets, but the chosen library is React. It is a component-based JavaScript library for making user interfaces. One of the main features of React is making components that have their own state [40]. When compared to popular JavaScript frameworks and libraries like AngularJS, Angular, Vue.JS, and jQuery, React was the most downloaded front-end JavaScript library in the past 6 months from node package manager, *npm* [41]. UI elements implemented with React are also easy to be added to just some parts of the web applications even if the site isn't entirely single-paged [8].

5.4 Dependencies

Before web application can be deployed and used, all the dependencies that the application needs must be installed or available for usage. IBM Knowledge Center mentions [42] following dependency types: transactional, service, IP, system and application to application dependencies. Beside of these dependency types, he actual web application and environment where the application is executed can have their own library and tool dependencies too.

All the dependencies that are needed to run an application with Node.js runtime environment are installed with Node.js installation package. Node.js dependency list contains different libraries and tools that are used in the core functionality of it. [43]

When making an application with *Node*.js and *npm*, dependencies needed by the project are stored in *package.json* -file. Npm uses this list to install all the application specific libraries and tools needed by the project or the libraries in the project. Version numbers of dependencies can be set to an exact version or to the latest available version. Dependencies are installed when running command *npm install*. [44]

The designed web application has dependencies to few other web applications in the same network that provide an API to get and manage data. These services are License-API, License-Management-API, Reporting-API and Login-API. The environment was described in chapters 4.1.1 and 4.1.2. Application also has a service dependency to Redis in session management. All these services need credentials from the using application. Credentials are stored in a configuration file that must be set manually by system admins for security reasons.

5.5 User interface and main features

User Interface is in very important role when designing a web application. Different kind of web application need different kind of approach to User Interface Design. Rossi et al. divided [23, p. 79] web pages to navigational, informational and data input types. Designed solution is mostly navigational and informational, so it needs to have a clear navigational structure and the provided information needs the be shown in understandable format. However, for some user groups and in some user cases, the application is a data input typed, so the user interface should provide data entry fields and clear descriptions for them.

McKay introduces “UI is Communication” -design principles in his book [45]. He mentions that user interface is a conversation between users and the system. Used system should always aim to do exactly what the user wants. This should guide the UI design process. He also says that all the offered tasks in UI should be explained very clearly and in a same way it would be explained in person. The other principles are that UI elements should be evaluated on how well they communicate, communication should always be polite and smart, and it should feel like a good conversation with a natural and friendly person. [45]

Design of the User Interface in this web application aims to be clear, well-structured and it should be able to communicate in a way that user feels like being guided and instructed to do the wanted tasks.

5.5.1 Website structure

Website structure can be introduced with a navigation diagram. Figure 10 shows the navigational structure of the website. The most relevant navigational links and relations are presented in the diagram. Website contains many shortcut links that are not considered because they are not relevant to describe the structure of the site.

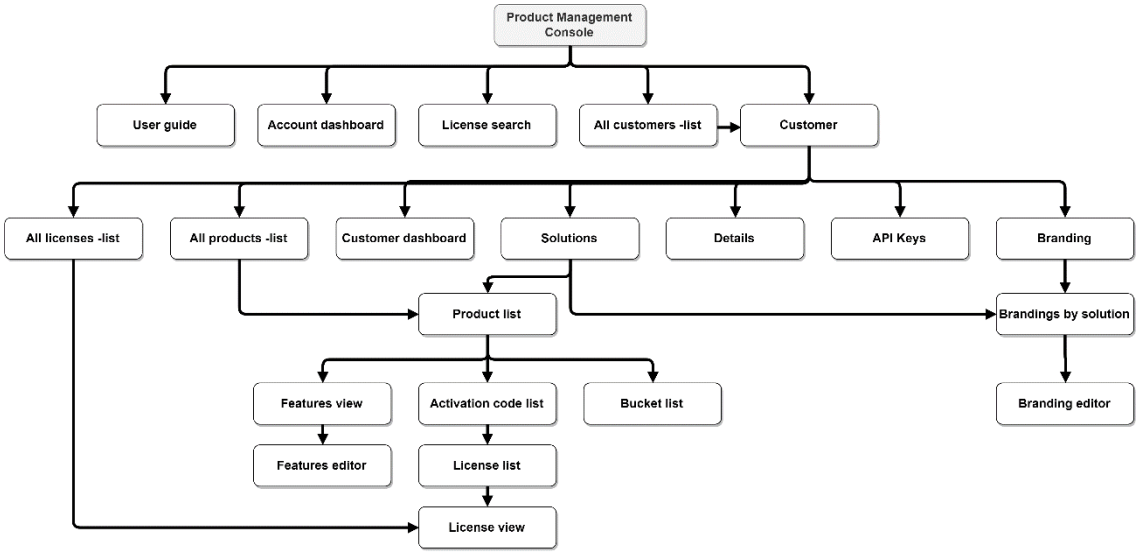

Figure 10. *Navigational structure of the website*

As the figure 10 shows, first level pages like admin/account dashboard, license search, customer listing and customer details are accessible from all the levels, if user has access rights to open them. Access rights to different pages are defined in chapter 4.5.3. From customer page, user can access to all licenses -list, all products -list, customer dashboard, solutions, details, API keys and branding. More specific licensing information can be seen by navigating from products list to activation codes, buckets, features and licenses. Branding editor is accessible from branding by solution -view.

The basic layout of the designed user interface is mainly formed from navbar, sidebar, footer and content. Navbar contains links to pages that are available always and are not related to any customer. If customer is selected and currently handled, sidebar is shown, and customer specific actions are offered. Basic layout of the designed UI is presented in figure 11.

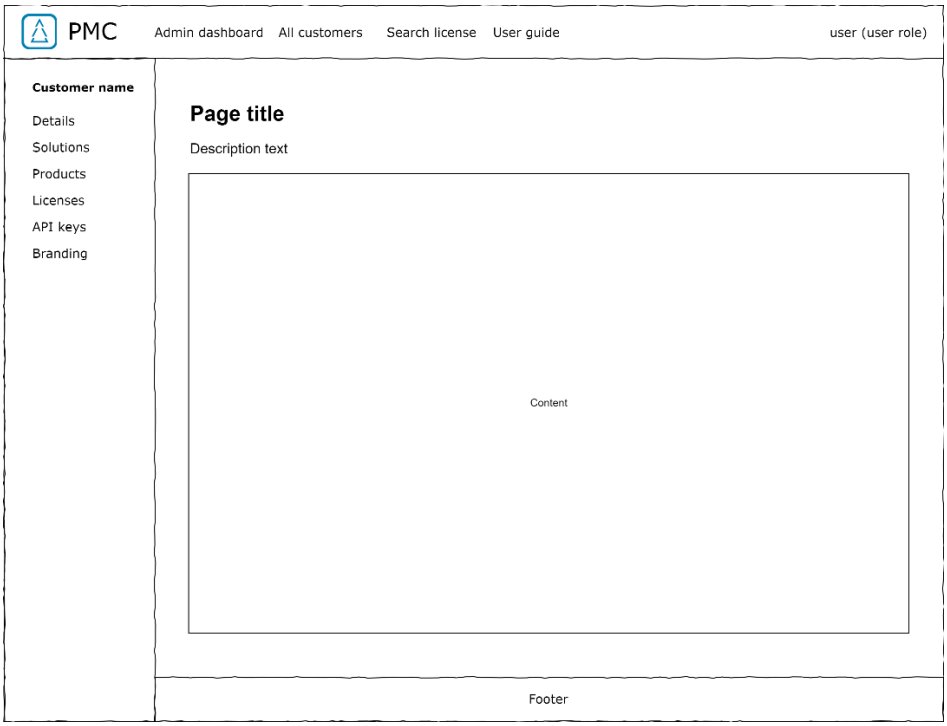

Figure 11. *User interface layout*

Navbar also contains dropdown button on the right side of the page as figure 11 shows. Here user can logout or change the language of the website. There is also a link to account settings page. Footer contains basic information of the company and is same as in other company's websites.

5.5.2 Access rights to different pages

Different user roles have different access rights to website's pages. Access control of the website's view is shown in table 2.

Table 2. *Access rights to different pages*

Role / page	Admin	Support	Salesman	Reseller	User
Customer dashboard	X	X	X	X	X
Account dashboard	X	X	X	X	
Customer ownership management	X	X			
Search license	X	X	X	X	
All customers	X	X	X	X	
Customer details	X	X	X	X	X
Solutions	X	X	X	X	X
Products	X	X	X	X	X
API keys	X	X	X		
Activation codes	X	X	X	X	X

Buckets	X	X	X	X	X
Licenses	X	X	X	X	X
Features editor	X	X	X	X	
Branding editor	X	X	X	X	X

As table 2 shows, admins and support team members have access to all possible views. Salesman and resellers have also wide rights to different pages because they can manage and search their own customer. Normal users have a bit more limited access because the management is restricted to their own items. Even though many user groups have almost the same access rights, there is a big difference on what data is shown and what actions users can perform. User access control and rights are introduced more specifically in chapter 4.5.2.

5.5.3 Customer dashboard

Customer's dashboard is the first page users see when they log in to system. The general purpose of a dashboard is to show user the most essential information from all the mass data. For example, in business context it could be number of sales per month for the past 12 months and the data could be in showed as a line chart. This would give the user a clear picture of the state without need to explore all the data. Dashboards usually contain many kinds of essential information entities. [46]

Customer dashboard is same for all user groups and the data is always shown from one customer at a time. For normal users, who have access to only their own customer details, this is the only dashboard they have and only their information is available. Users who have access to other customers' details, can view their own dashboard or navigate to one the available customers' dashboard. Customer dashboard contains relevant customer information like list of expiring licenses and product statistics. Users will also see list of most common actions and shortcut links to execute these operations.

5.5.4 Account dashboard

Second dashboard in the application is the account dashboard. In this view, admins, support members, sales and resellers will see only information that concern them. Many admin tools, like *JIRA* have customizable dashboards where users can configurate that what kind of widgets they will see [47]. In this thesis, dashboard's widgets are defined and shown by user group.

Account dashboard is divided to three areas. Quick search component allows user to search from those customers that the user has access to. Second component contains list of expiring licenses from the customers that the user can manage. Third component contains different kind of product statistics from own customers.

5.5.5 Search page

Support and sales users who are responsible for customers and get support requests, need to find the handled license quickly and with a small amount of information. There is sometimes available only some small pieces of information, like part of the address of the customer's store or few characters of the license id. Search feature accepts all kind of input and gives results in readable and compact format. Search view with licenses and parent resources is presented in figure 12.

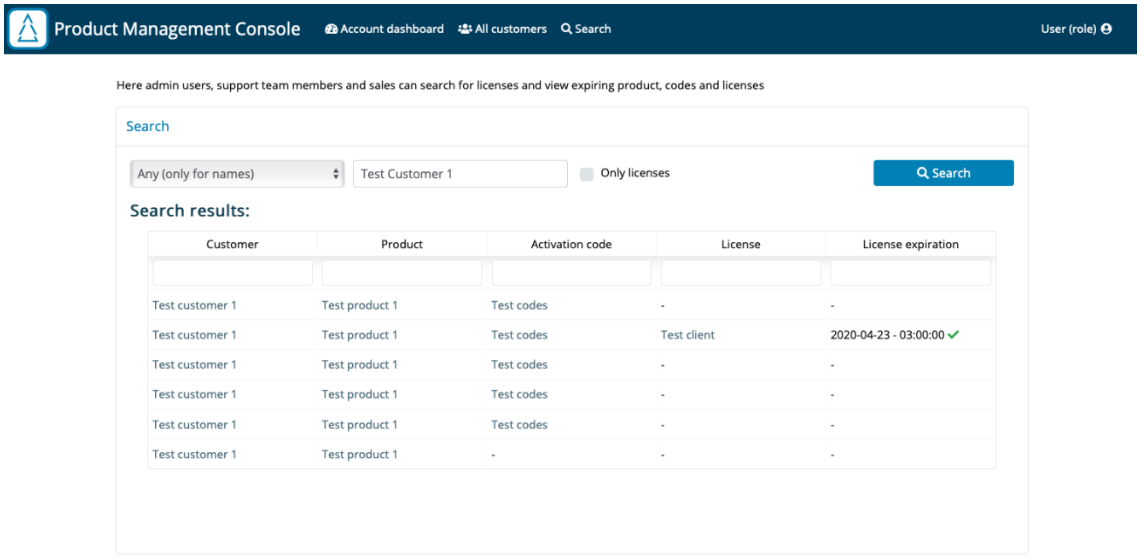

Figure 12. *Search page with links to resources*

As figure 12 shows, admin can search with any name which can contain customer name, product name, activation code name, license name, store address. There are also search options for different identifiers like customer id, product id, activation code and license id. Search accept partial inputs and is case insensitive.

Usually the main target in search is to find the correct license and see all the details for product and list its enabled features. Search results are shown in two different formats, default format shows license results with parent activation code, parent product and parent customer. All the parent resources also contain link to resource page. License results have a link to license view and expiration date with colourful sign indicating how soon

the license expires. The second format is a plain license list with link to license and same kind of expiry date indication.

Search page is implemented with React and has no page reload. However, it writes search field and keyword to URL as GET-parameters so that search results are linkable and won't disappear if page is reloaded. Search results division is scrollable so that search terms can be edited at any point of the search process.

5.5.6 All customers

In the old Product Management Tool, customers were listed in the front page and the navigation started from there. Now, when the tool is used also by the customers, only those who have access to other customers, can access this page. This means that admin users and support team members can list, search and manage all the customers from this page. Salesman and resellers see a shorter list which contains the customers they can manage.

New customers can be added from this view, if user has access right to do so. When user presses "Add new customer" -link, a modal opens and offers fields to add user information. Customer listing page also has search options to look for certain customer. Search field uses values from all the columns in the table. Sorting can be made for all column in both directions. Customer listing page is shown in figure 13.

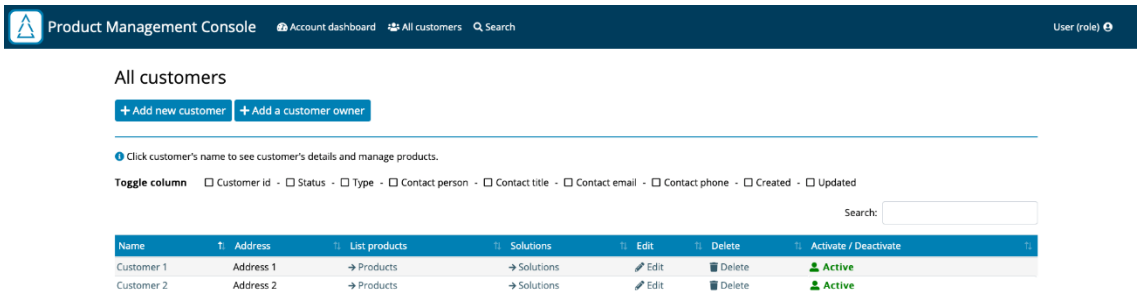

Figure 13. *Customer listing page*

Like figure 13 presents, shown customer information by default is name, address and contact information. All the other possible fields can be toggled available from toggle bar and are included in searches even when they are hidden. Users are offered with links and actions to customers. User can navigate to customer's products and solutions via separate links. Users can edit customer details and when pressing "edit" button editing

modal opens. Some users can also remove customers and when delete button is pressed, a confirmation dialog opens. Customers with products can't be deleted.

5.5.7 Customer details

Customer details -page shows customer's own information and lists all the customer contact addresses. User can update customer's details by pressing "Edit details" -button and update form opens in a dialog. Current address details are listed here. User can add new address detail set from separate dialog. Customer details -page is shown in figure 14.

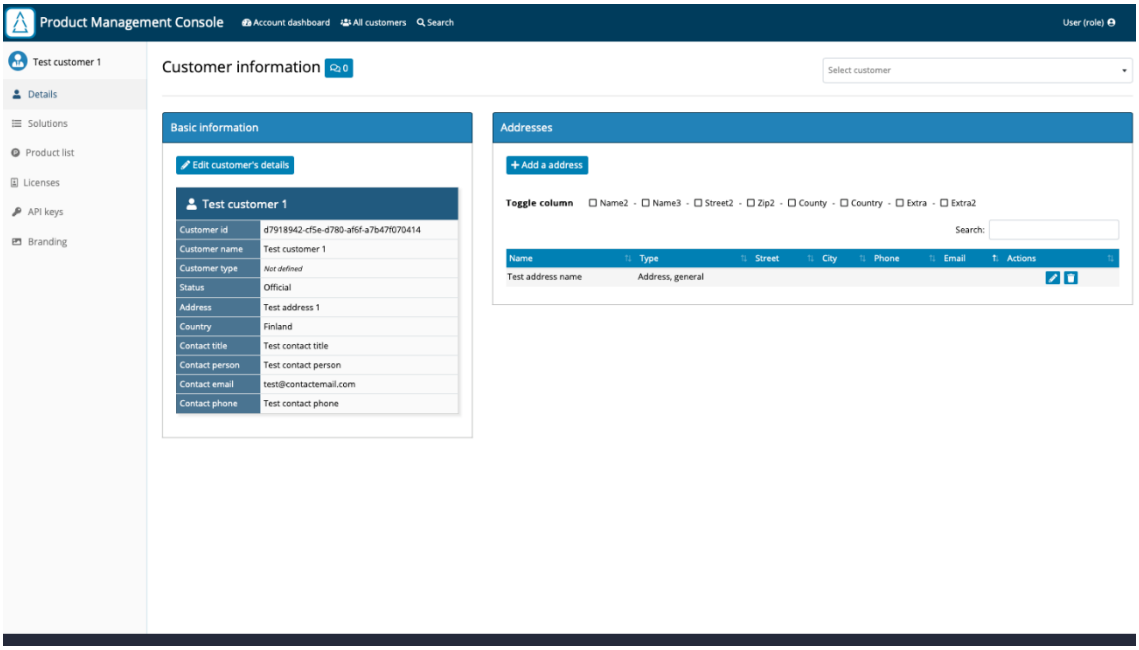

Figure 14. *Customer details -view.*

Figure 14 shows the basic information and addresses in own cards. All the actions related to that part are inside the card. There is also a possibility to change the handled customer from a dropdown in top right corner, if user has access to other customers.

5.5.8 Product, activation code, bucket and license managing

Managing license related items is a basic feature from the old management tool. User with enough access rights, like admins and support team members can make basic CRUD- operations (create, read, update, delete) for all these items, except license objects, which are generated when the software is registered using activation codes

Products are listed in a table and the most relevant information is shown in the columns. Table can be sorted from wanted column and rows can be filtered using a search, which filters using values from all columns. User can also toggle hidden columns available. For

users who can manage items, action buttons to edit and delete items are shown. For every product, there is also links to product's features, activation codes and buckets. Detail page of one product lists all the activation codes. Product details page is shown in figure 15.

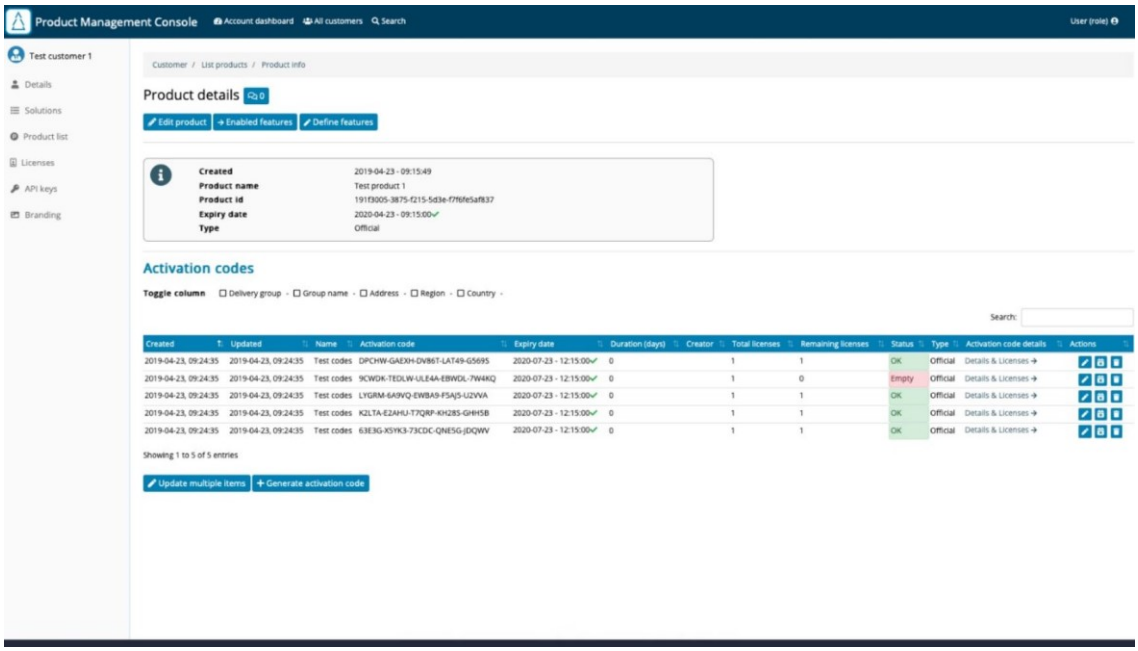

Figure 15. *Product details and activation code listing*

Page shown in figure 15 presents product details and all the activation codes of the product. Activation codes are in a same kind of table as products are. All the sorting and filtering features are the same so that the system is concordant when it comes to data tables. Visible details in activation code items contain basic information like name, code, expiration dates and amounts of total licenses and licenses left with the code. Every row also contains link to license listing. For users who can manage these resources, there are also buttons for modification, invalidation and deletion.

In license lists, all the activated licenses are listed for certain activation code. Like in the previous chapter, also license listing has the basic details of the parent item, which is in this case the activation code. For admins, there is a button to revoke a license. A confirmation dialog is opened when performing this action and a revocation reason is asked from the revoking user.

It is sometimes necessary for support team members to modify multiple items at the same time. There is a possibility to select multiple items and edit them simultaneously in all these lists. Mass edit can be done by clicking wanted items and pressing "update multiple items" and inserting updated information to dialog. When information is filled and saved, it is updated to all selected items.

5.5.9 Branding editor

Moving branding creation to web application and giving the customer an opportunity to create branding by themselves is one of the biggest changes to previous processes in this thesis. The basis in planning the branding editor was to make the architecture support expansions. At this phase of development, customer can edit their logo used in the PiceaServices, PiceaOne and PDF branding or customize their PDF reports by selecting shown fields. But in the future, it could be possible to edit many style options and elements, so the architecture should support adding a new editor view easily. Because of this, the editor was implemented with React, which allows easy usage of components and makes the user interface interactive. Branding editor is shown in figure 16

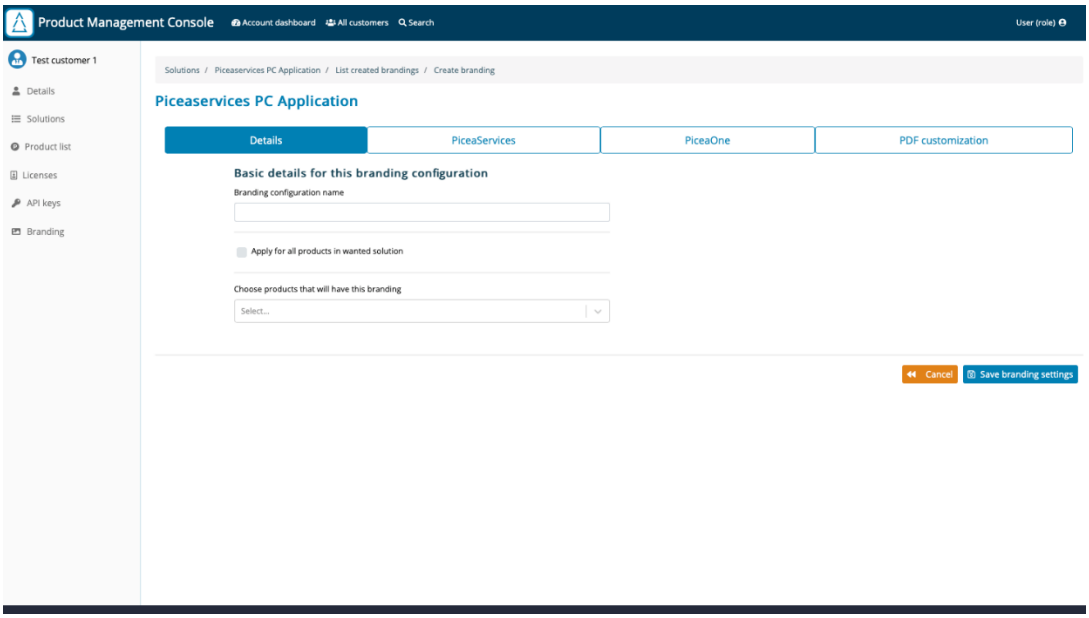

Figure 16. *First view of the branding editor.*

Figure 16 shows the first view of branding configuration. In this view user must give the basic information for the configuration. This contains the name of the configuration, selecting the products that it influences or if it is generic for all products in this solution.

Next views are for adding and previewing used logos. Depending on solution, there are images for PiceaServices PC App or PiceaOne mobile application containing the inserted logo image so that the user can see what the user interface would look like with the logo. Logos can be in PNG-, SVG- or GIF-format and must be less than 128kb.

When the logo is defined, user can move to PDF customization. The idea of PDF customization is to allow users to define that what fields are shown in the PDF reports retrieved using customer's products. PDF report is given for the end-user in the stores after the operations made with Piceasoft's products. By default, every field is enabled. User can disable fields by clicking wanted checkboxes off. User can also download a preview

PDF file with currently made configurations for wanted report type and see what an actual PDF report would look like. No pre-saving must be done when downloading a preview PDF file. There is also possibility to add a PDF-specific logo that would be used only in the reports.

5.5.10 Product features editor

When creating a new product for a customer, user inserts the basic information, like name and expiration date. In this phase, only the default features are added for the product. After a successful creation, user is redirected to features editor page. Before describing the features editor, it is necessary to explain the previous feature management and the newer structure.

In the old model and tool, features were shown and managed as a flat list with no relation between them. Now there is a new structure for solutions, services, features and application settings, so that user can right away understand connections between them. Main level in feature handling is solution, because different solutions have different services to be enabled. There can be multiple different solutions and common application services are handled as one solution. All the solutions can have *n* number of services.

One service describes a service in Piceasoft's solution. For example, in PiceaServices PC Application, one service can be "Switch", which is for transferring content from mobile device to another. Services have predefined identifiers and can be set enabled by default. Services can have *n* number of features enabled.

One feature is a smaller application feature that is used in defined service. If we use the previous example of PiceaSwitch -service, one feature could be "Full screen" which would allow usage of full screen mode in this service. Features can have *n* number of application settings defines as "details". Application settings, also known as feature details, define even smaller feature specific things, like maximum number of devices allowed. Features editor -page is shown in figure 17.

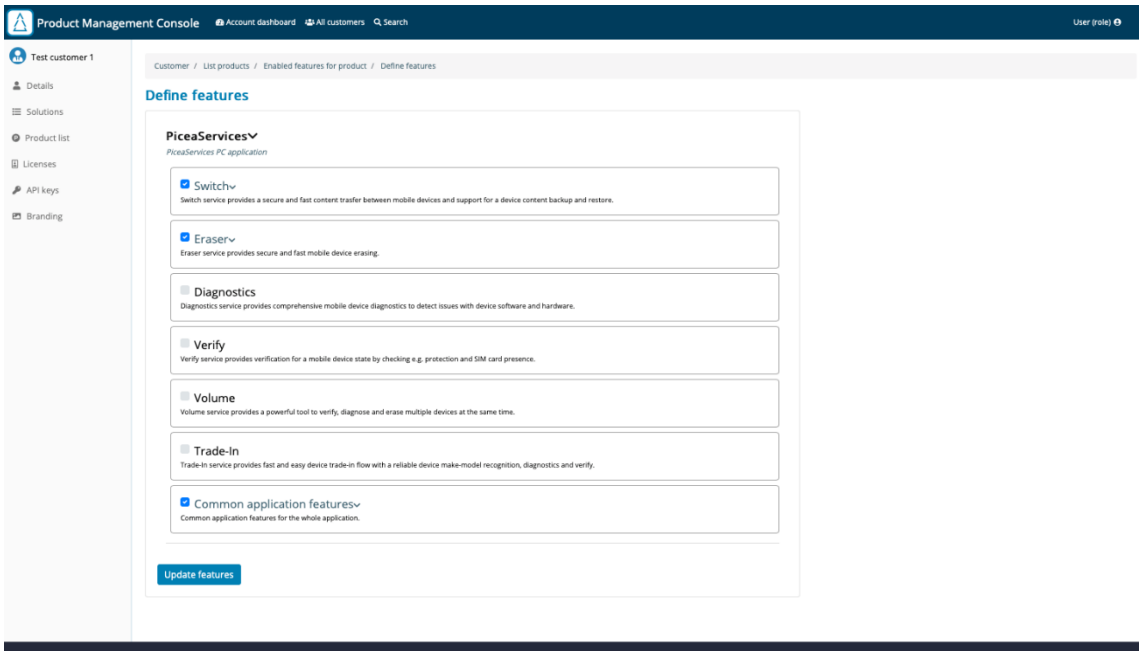

Figure 17. *Feature editor and services for PiceaServices PC Application*

As figure 17 shows, services are checkboxes that can be enabled or disabled. Features are enabled with same kind of checkboxes as services. Application settings can be checkboxes, text or numbers. User can expand and collapse service or feature by clicking the title.

5.6 Security and access control

5.6.1 User groups

There are five user groups specified for the application: admin, support, sales, resellers and basic users (customers). Table 3 shows user groups of the system.

Table 3. *User groups of the system*

Level	User group	Details	Data access	Internal / external
1	Admin	Administrates the whole system and has access to every resource	All	Internal
2	Support	Can manage all the customers	All	Internal
3	Sales	Can manage customers who belong to salesman. Can create new customers	Own customers	Both, mostly internal
4	Reseller	Can manage own customers	Own customers	External

5	Basic user (customer)	Normal user is customer's representative. Can manage and view own resources. Has limited editing actions and resource details.	Own	External

Table 3 shows also the data access level and if the group is for internal of external users. The most active user group of the system is company's support team members, who handle customer issues related mostly to licensing and product features. The basic users are representatives of the customers, who are responsible to handle customer's branding, settings, and making configurations to products.

Users with some extra privileges are sales agents, who have access to their customer's settings and products. Resellers are quite like sales agents and they can handle customers who are particularly set as their customers. Difference between sales agents and resellers is that sales agents are mostly internal and resellers entirely external users. Admin users have access to manage the whole application and data in it.

5.6.2 Access control

Access control means checking if user can do something or have access to specific resource in application. In most of the web application, there are multiple different user roles with different privileges, so access control is in very important role improving security of the application. [48] Designed system has 5 different user roles with different access rights, so having a secure access control design is very important.

OWASP has listed four main access control types: Role Based Access Control (RBAC), Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Permission Based Access Control (PBAC). [48] From these suggested options, Permission Based model suits for designed solution, where is different kind of resource types available for reading, modifying, deleting and creating. OWASP suggests PBAC model if there is a possibility to classify resources and define specific grants for all actions [48]. PBAC model can be used when there are user groups made and permission grant can be given to a whole entity at on. This makes the access control system more maintainable if there is a lot of users who have access to the web application. [48]

OWASP has mentioned some access control attacks types that should be considered when designing a web application. In vertical access control attack, user tries to do actions or see resources that are out user's privileges. For example, as a normal user trying

to perform an action that only admins can. Horizontal access control attacks aim to get access to other users' resources with same user role. [49]

The access control in designed solution is based on Permission Based Access Control -model. All the resources in this system are defined in a way shown in program 1.

```
{
  "product": {
    "read": {
      "admin": true,
      "support": true,
      "salesman": true,
      "reseller": true,
      "user": true
    },
    "create": …,
    "modify": …,
    "delete": …,
  }
  …
}
```

Program 1. *Resource permission control*

As program 1 shows, the access control permission is handled in JSON format. The structure is defined in order: resource, operation, role. This format supports generation of user right matrixes for editing rights or only listing user rights for testing purposes. User right matrix is introduced in figure 18.

Product

	Admin	Support	Salesman	Reseller	User
read	X	X	X	X	X
create	X	X	X	X	
modify	X	X	X	X	
delete	X	X			

Figure 18. *User rights matrix for resource "Product"*

From figure 18 we can see that the user rights can be expressed in a more visual way that could also be a template for dynamic access control management UI. This kind of table helps security and testing team to ensure, that access control system works like it should. In application logic level, the access control is checked with a function, which takes resource, permission and user-object as parameters and returns true if access is

granted. Access control is also implemented as a middleware to check if request can go to wanted controller function.

```
router.put('/customer/:customer_id',
 piceaPassport.checkAccessMW("customer", "modify"),
 CustomerCtrl.put_update_customer);
```

Program 2. *Middleware check for every resource and action*

In program 2, application gets PUT request to URL "/customer/:customer_id". Middleware function *checkAccessMW* checks if the request can be passed to the end-point controller function. If access is not granted, application responds with correct status code and error message depending on what kind of request was received.

5.6.3 Possible security threats and solutions avoiding them

The Open Web Application Security Project, OWASP, has made a top ten list of the most critical security threats of web applications. The latest list was made in 2017 and these threats are picked up as security topics in this section. [50] This list is mentioned in the system requirements in section 2.3.2 and is used as a basis for designing security of the application.

SQL Injection

SQL Injection attacks are made by inserting SQL commands as inputs so that the receiving system would execute these commands in the database system. This is very common attack type in web applications with a connection to database. [51] Designed system does not have a straight connected to database, so the responsibility to validate, filter and parameterize inputs is moved to those services who offer the interface to manage data.

Broken Authentication

Authentication consists of confirming user's identity and in web application is made when user logs in. If user credentials are correct and user has access to web application, a session is made for the user. Designed solution doesn't have own user database. User authentication is made against common Login-service, which is used by all other services too and is well tested. Having stable and secure login service in the back-end makes a threat of having broken authentication very unlikely.

Sensitive Data Exposure

This area is very critical in licensing system. It is important, that users can't access data that they don't have permission to. In designed solution, data access checks are made for every request by using specific middleware function. Having only one middleware for making data access control makes it more maintainable.

XML External Entities (XXE)

If application accepts XML as input and is parsed by an XML processor, may web application be vulnerable to XXE attacks. OWASP recommends usage of simple data formats, usage of updated XML processors, positive server-side validation to avoid XXE attacks. [52]

Designed solution uses mostly JSON data format when sending or receiving data from internal or external sources. XML processing is possibly executed in needed libraries, but they are kept updated and found vulnerabilities are monitored from them.

Broken Access Control

Broken access control affects also on sensitive data exposure and is carefully designed and maintained. Designed system uses permission-based access control, where permissions are granted to resources according to user group. User groups are attached to user information and are retrieved from Login-service. User groups are managed only by system administrators.

System's automatic tests also include access control testing, where all the tests are executed with users from all user groups. Test results are compared to expected results and administrators are alerted if tests fail and user can access to resource when access should be denied.

Security Misconfiguration

The system has centralized configuration file which has strict permissions in deployment environment. It is always manually set and has only up-to-date template available. All the security related configuration values are set there and is checked in system startup.

Express.js documentation proposes [53] usage of security package *Helmet*, which is a set of middleware functions that set HTTP headers related to security issues. It handles following headers: Content-Security-Policy, X-Powered-By, Public-Key-Pins, Strict-Transport-Security, X-Download-Options, Cache-Control, Pragma, X-Content-Type-Options, X-Frame-Options and X-XSS-Protection. Helmet is used in designed application to handle security-related HTTP headers.

Cross-Site Scripting (XSS)

Most of the XSS attacks can be prevented with encoding all the untrusted data which contains mostly user inputs. *React*, which is used in some parts of the application, escapes XSS automatically. [54] Most of the HTML code in the application is rendered in server-side and all the untrusted data is escaped.

Insecure Deserialization

According to OWASP [55] the safest way to avoid insecure deserialization is to deny serialized objects from sourced that are not trusted or to allow only specific data types. Serialized data from untrusted sources are avoided in this system.

Using Components with Known Vulnerabilities

Node package manager (npm) is used in installing and maintaining libraries and components in Node.js projects. Npm installs and updates libraries that are defined in project's *package.json* -file. Npm also maintains information about found vulnerabilities in packages and reports if the project contains any. Running command *npm audit* in project path gives security report and tells the amount of found vulnerabilities in the project's package list. [56] All the components used in the solution are installed using npm and therefore checked for known vulnerabilities.

Insufficient Logging and Monitoring

User activity monitoring is important, if web application has critical information that can be managed or viewed, and users can have privileged access rights. Activity monitoring is often made by making audit trails by logging user actions from the time when user is logged in. Auditing helps to find out that which user has done critical operations or who have accessed some sensitive data [57]. OWASP suggests making audit logging from all successful and failed logins and important transactions and store it to some centralized location where it can be monitored by administrators .

Only administrators have access to audit trails. Audit trails are chronological, which means that logging includes timestamps. Logging also should include user information, action information, result of the action and information if some resource is changed.

In this environment, there is a ready Audit-API that receives audit log from all the applications. API needs following information:

- Application id, in this cases system's own id
- User who has done the action
- Event type, which describes shortly handled resource and action, for example "*ActivationCodeDeleted*"

- Event details, which has information about the resource, like "*Product id: *uuid*; activation code: *code**"
- Severity, which tells that how important this audit logging is. By default, severity is set to the lowest level, *info*. The highest severity level is *critical*.

User action logging is not made on every request to the tool. If user is listing some trivial non-sensitive information, logging would just make auditing more difficult by adding log lines. Logging is made when user makes access to sensitive data, modifies existing data or deletes items.

5.7 Deployment

Current deployment process must be considered when designing deployment for a new web application. Currently deployment to production is made with blue-green strategy, where updated version is deployed for automated testing while the currently deployed stable version is still available for users [58]. If tests fail, deployment is rollbacked and stable version brought back to use on both instances.

System for deployment process is Gitlab CI/CD. Deployment to R&D environment is made every time commits are pushed to development branch in Git repository. Complete set of automated tests is executed for deployed version so that the environment is always tested. Deployment for production is made by making merge request to master branch. Request must be accepted and reviewed by person with rights for it. This process is made for all web systems and is a constraint for planning application specific deployment configuration.

Express documentation suggests using process managers when deploying Node.js applications with Express as the main application framework. Using a process manager ensures that the application is restarted immediately if it crashes and improves the availability of the service. The most common process managers for Node.js -applications are StrongLoop Process Manager, PM2 and Forever. [59]

Proposed process manager for system is PM2. It is lightweight and has all the needed features. StrongLoop is no longer in active development, so it would not be a safe choice, even though it has some good features too [60]. Forever is mostly used for automatic restarts of applications and lacks many features that PM2 has. Deployment diagram is shown in figure 19.

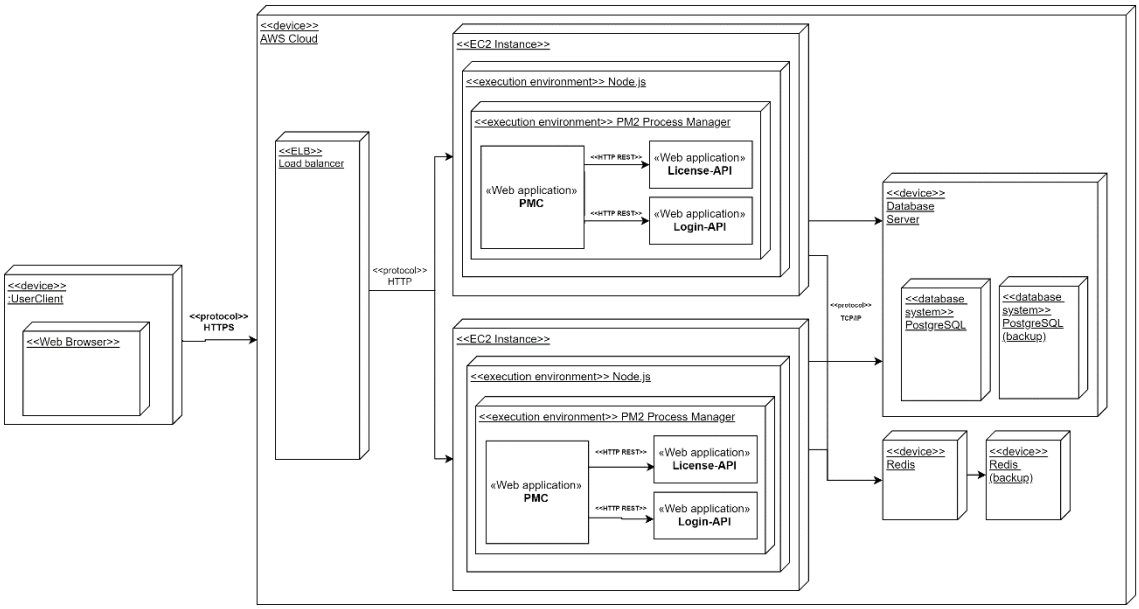

Figure 19. *Deployment diagram of designed solution (PMC)*

Figure 19 shows the deployment environment with the dependencies and protocols used. Web applications are executed inside two EC2 instances in AWS Cloud. Both instances contain Node.js execution environments which execute PM2 process managers. PM2 instances are executing and managing applications processes.

PM2 supports many things that are useful in present and possibly in the future. Now load balancing is handled with *Elastic Load Balancing (ELB)*, but PM2 supports it too. PM2 supports running applications in cluster mode, which means that one application can be scaled by executing it in multiple processes. Number of clusters is usually set to equal the number of available CPUs in the server. Using cluster-mode allows having zero downtime in application restarts when applications need to be restarted, because processes are reloaded one at a time. [61]

6. EVALUATION AND DISCUSSION

6.1 Requirements

Functional requirements of the systems were listed in section 2.2. Designed and implemented system contains almost all the features that were wanted in the requirements. User guide was not included in this phase. This feature should be made as maintainable as possible for the support team and implementing the feature did not fit in this phase of development but will be included in the next release. Many of the non-functional requirements are hard to evaluate before the system has been taken in production use.

All the existing and wanted functionality from the old product management tool was implemented to the new system. Most of the extra features that would have been needed already in the old tool were implemented in this system. The most important new features were mass editing and improved search page.

The system supports all the user groups that were specified. Most of the access rights for different resources or actions were not defined very specifically in the requirements and have been changed during the implementation process.

6.2 Architecture and technology choices

The most important architectural design choices for the solution were selection of web application type and the application architecture. Web application type was eventually chosen to be MPA with some parts made with React. This was a good decision from schedule point of view, because developing the user interface and the business logic entirely with React could have taken more time. It also suited quite well for the nature of the application and the user experience remained good even though there were page reloads. React fitted well for those parts where interactivity was needed.

New licensing API was implemented at the time as the web application for management. Application uses this API for all data management and this saved a lot of time and effort in the development of the management system. All the database queries are made and maintained by that service.

6.3 Comparison to current management tool with usability evaluation

This chapter contains a usability research where old management tool and designed system are compared from usability point of view. Purpose of this research is to get usability data which can be used to evaluate the usability of the designed solution. First the research method and tasks are defined. After the research is made and necessary data collected, results are shown and briefly analysed.

6.3.1 Research and usability testing methods

Usability research data is collected with observation. The role of the researcher in the observation is observer as participant, where the researcher does not take part in the tasks and the identity of the researcher is known [62, pp. 293-294]. Used usability testing method in observation is performance measurement method and the collected data from user tasks is quantitative [63]. This method suits well for this kind of usability research where data is collected from two different systems that support same features for predefined tasks and can be compared.

Selected users for this usability research are company's internal users due to sensitive data that management tools handle. Selected users haven't used either one of these applications but understand what the licensing structure of the company is. Users will start tasks 1 and 8 from navigational front pages and other tasks from the page where the previous task ended. Observer won't guide or talk to user unless user does not know something about licensing that should be known.

Collected data in performance measurement will contain both elapsed time and number of navigational actions that were needed to perform the task. Elapsed time will be rounded to closest second. Recorded time will start from the point when the user starts to perform the task and is ended when the conditions of executed task are fulfilled. Number of navigational actions means navigations to new pages, opened dialogs or clicked buttons.

6.3.2 Tasks

Selected users will be asked to perform a list of tasks that are possible with the current system and the designed system. Tasks are written in a way that users will understand them unambiguously and contain all the necessary information to perform the tasks. All the tasks start from the from page that is shown after login and is the front page of the

website. Tasks relate strongly to licensing management or information searching. Features that are not possible in current tool, like branding and PDF customization, are not included. Specific values that are needed in testing are written and easily copied in a text file which is open in the used computer. Tasks are defined in table 4.

Table 4. *List of tasks used in usability test*

#	Task
1	Create a new customer
2	Add new billing address detail for created customer
3	Change contact person of created customer
4	Create a new product for created customer using any input data.
5	Enable transfer / switch feature for created product
6	Generate five (5) activation codes for created product with defined values
7	Change types of created activation codes
8	Invalidate/expire created activation codes
9	Get the number of activated licenses for defined customer and product
10	Find product and feature details of client id with partial information

Table 4 shows the number of the task and a short description of it. If the task requires more information or some input values filled, they will be found from the task material which is opened in the computer. Task is filled when the result is accepted by the researcher and the user acknowledges that the task is done.

6.3.3 Results

Five users participated in this performance measurement. Collected data from performance measurement is numerical data. It can be categorized as ratio data because we can calculate relative differences between both elapsed time and performed steps. Elapsed time is continuous data and number of performed steps is discrete data. [62, pp. 418-419]

Both elapsed time and number of performed tasks should be shown with statistics. Sanders et al. proposes calculating mean or median values for both continuous and discrete data. Median can be better when data can contain some values that differ greatly from the other values. [62, p. 445]

Table 5 shows measured time from PMT and PMC with mean and median values of collected data. Table 6 contains median and mean values of performed steps for both systems. Both tables also present the percentual differences of calculated values.

Table 5. *Median and mean values of elapsed time for each task*

Task	PMT median T(s)	PMC median T(s)	Median time difference	PMT mean T(s)	PMC mean T(s)	Mean time difference
1	20	11	-45 %	23	15	-34 %
2	32	36	13 %	89	37	-58 %
3	56	12	-79 %	56	13	-77 %
4	21	19	-10 %	21	19	-10 %
5	12	8	-33 %	12	10	-19 %
6	64	50	-22 %	81	44	-45 %
7	49	35	-29 %	49	38	-23 %
8	46	44	-4 %	53	37	-29 %
9	44	37	-16 %	52	62	19 %
10	187	28	-85 %	220	59	-73 %
Total	531	280	-47 %	654	333	-49 %

Table 6. *Median and mean values of performed actions for each task*

Task	PMT median actions	PMC Median actions	Median action difference	PMT mean actions	PMC mean actions	Mean action difference
1	2	2	0 %	2	2	0 %
2	4	5	25 %	9	5	-47 %
3	5	2	-60 %	5	2	-54 %
4	3	3	0 %	3	3	13 %
5	2	2	0 %	2	2	-9 %
6	4	2	-50 %	5	2	-52 %
7	10	10	0 %	10	10	0 %
8	10	10	0 %	10	10	2 %

9	5	4	-20 %	6	6	4 %
10	20	5	-75 %	22	5	-77 %
Total	65	45	-31 %	73	48	-35 %

Table 5 shows percentual differences between PMC and PMT for median and mean values. Elapsed time for performing task is lower with PMC in all tasks. Total time difference of median times for completing all the tasks was 47 percent lower with PMC.

As table 6 shows, many tasks required same number of actions in both web applications. Biggest differences were in user customer detail editing, activation code generation and license searching. Median number of steps in searching a license was 75 percent lower when using PMC. Figures 20 and 21 present performance measurement results in bar charts.

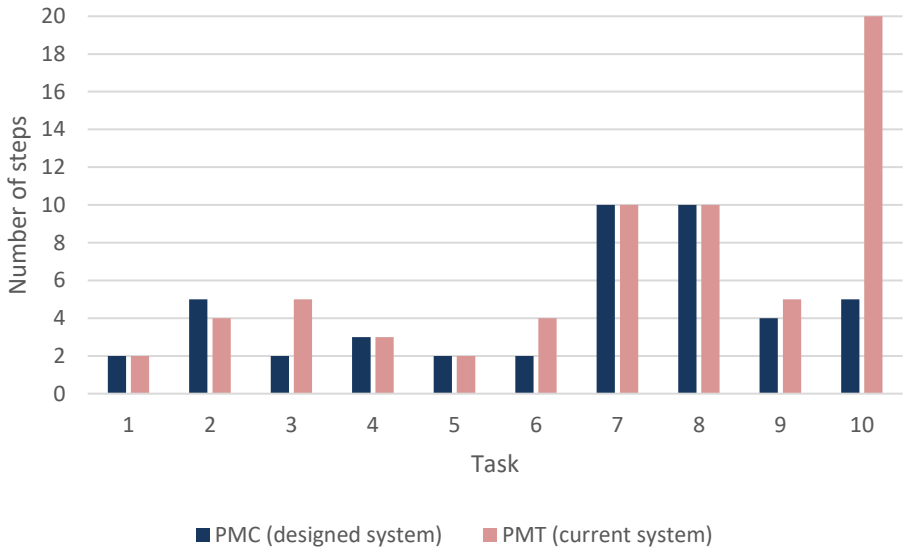

Figure 20. *Median number of steps by task*

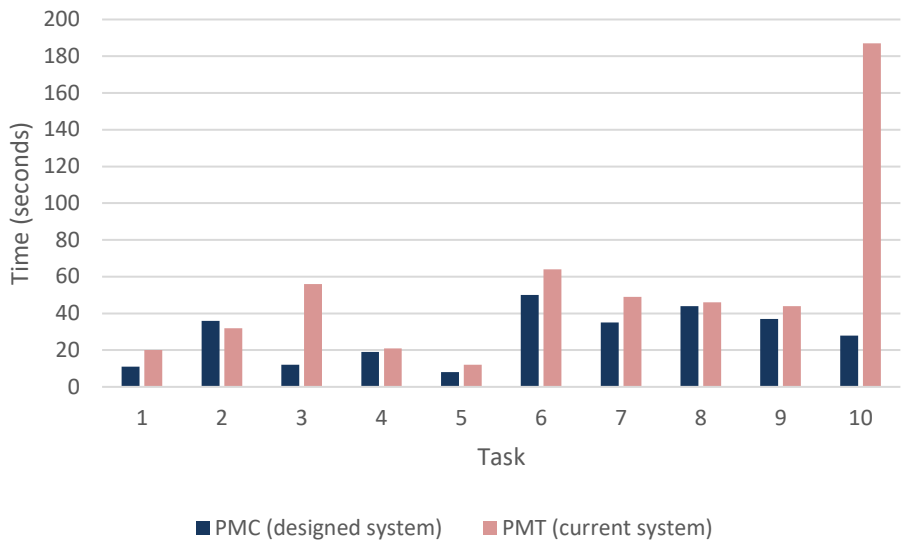

Figure 21. *Median elapsed time by task*

Figure 20 presents median number of performed steps for every task. Results using PMC are shown with blue colour and results using PMT using red colour. Figure 21 uses the same colours for both systems, but the values present median elapsed time for executing single tasks. Median values were chosen for these bar charts because using median can eliminate extreme values that might have been caused by external reasons.

6.3.4 Discussion

The results of the usability performance measurement were mostly expected. The old system had some known usability issues that were considered when designing the user interface of the new system and this was shown in the results. Especially results from tasks where single resources were edited or searched were very positive. Search logic in the old system was very clumsy and this appeared as frustration when users performed the task . Some tasks were a bit faster with the old system because they were very simple in the user interface of it. The new system had a bit more options in the user interface and it took a bit longer for users to notice the right action.

However, there were also some results that were not expected. Tasks 7 and 8 could have been made with new mass editing feature using PMC, but users weren't apparently guided to use this feature well enough in the user interface and they edited all the items separately. This resulted as a same number of performed actions but could have been performed faster if user would have been guided better.

Some tasks were navigationally quite similar using both systems due to same kind of licensing resources hierarchy. This might have an effect on the test results because users could remember that how some tasks were performed with the previous system even though the user interface and actions were different.

Overall results of the performance measurement show that the usability of the designed tool is better than the old tool from performance aspect. However, the amount of test users was relatively low because of the constraints for the research. To get more reliable results of the usability, there should be more test users and more tasks. This could be included as a regular method to improve the usability as the development of the system continues in the future.

6.4 Future development

This design and development phase of the web application included only the parts that were specified in requirements of this thesis. In the upcoming releases, the development of the project continues, and new features will be added. The design of the application allows easily adding new functionality in both back-end and front-end. However, there are some parts that should be investigated if the web application grows.

The user access control is now defined with static JSON-file. It is not changed very often, but it could be helpful if admin user could change access rights easily with user interface. Technically this would not be a very big task, but the storing of the access rights -data should be reconsidered and investigated.

Dashboards are currently shown according to user's group. In the future, user could select what kind of widgets would be shown in own dashboard. Current dashboard creation logic does not support this but adding this kind of functionality would be possible with some changes.

Feature definition -JavaScript-files that are generated from another project should be automatically included in this project. Now the files are included manually to the project files, but it could be possible to add a pipeline to synchronize the files to this project when they are changed in the original project.

7. CONCLUSIONS

The objective of this thesis was to design and implement a web application where products and licenses could be handled by many different user groups with different access rights. The system was designed to replace the current product management tool. It was important to retain the basic navigational structure from the current tool. There were also many new features that were included in the new system. New product management tool will help support personnel to fulfil requests related to licenses and will also give company's customers possibility to configurate their products by themselves.

The design of the web application was started by choosing system and application architectures that suited well for this kind of application. System was designed with multitier architecture and user interface with Model-View-Controller pattern. Second important choice was the selection of web application type. After comparing Multi-page applications, single-page applications and application types between them, the designed application was chosen to be a multi-page application with usage of SPA-oriented JavaScript libraries in pages where interactivity was needed.

The usability of the implemented web application was evaluated with a usability research that contained a performance measurement. Both new tool and old tool were compared with ten tasks performed by test users from the company. Results showed that the usability of the new tool was better in both elapsed time and performed actions.

Implementation of the system continues in the future and all the results from usability research will be used for improving the usability. Website will have much more new features that will be implemented in the future. Also, some of the current functionality, like dashboards and access control management are improved so that the system will bring as much value as possible for the users.

REFERENCES

[1] M. Mikowski, Powell J, Single page web applications: JavaScript end-to-end, Manning Publications Co., 2013, 407 p.

[2] Reqtest, Functional vs Non Functional Requirements, Available (accessed Mar 8, 2019): https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/.

[3] Agile modeling, Constraints: An Agile Introduction, Available (accessed Feb 24, 2019): http://agilemodeling.com/artifacts/constraint.htm.

[4] Techopedia, What is Web Development? - Definition from Techopedia, Available (accessed Mar 8, 2019): https://www.techopedia.com/definition/23889/web-development.

[5] Britannica, Client-server architecture, Available (accessed Jan 31, 2019): https://www.britannica.com/technology/client-server-architecture.

[6] TechTerms, Client-Server Model Definition, Available (accessed Mar 8, 2019): https://techterms.com/definition/client-server_model.

[7] G. Fink, Flatow I, Pro Single Page Application Development: Using Backbone.js and ASP.NET, Apress, 2014, 324 p.

[8] Facebook, Add React to a Website, Available (accessed Feb 16, 2019): https://reactjs.org/docs/add-react-to-a-website.html.

[9] Github, Turbolinks, Available (accessed Feb 24, 2019): https://github.com/turbolinks/turbolinks.

[10] NGINX, Introduction to Microservices, 2015, Available (accessed Apr 19, 2019): https://www.nginx.com/blog/introduction-to-microservices/.

[11] H. Schuldt. Multi-Tier Architecture, In: L. LIU, ÖZSU MT (ed.), Encyclopedia of Database Systems, Springer US, Boston, MA, 2009, pp. 1862-1865.

[12] Serverless Stack, What is Serverless? 2016, Available (accessed Apr 19, 2019): https://serverless-stack.com/chapters/what-is-serverless.html.

[13] C. Pitt. Pro PHP MVC, Apress, Books24x7, 2012, 500 p.

[14] W3C, HTML and CSS, Available (accessed Jan 30, 2019): https://www.w3.org/standards/webdesign/htmlcss.

[15] A. Freeman. The Definitive Guide to HTML5, Apress, 2011, 1051 p.

[16] Techterms, HTTP Definition, Available (accessed Mar 9, 2019): https://techterms.com/definition/http.

[17] Techterms, HTTPS definition, Available (accessed Mar 9, 2019): https://techterms.com/definition/https.

[18] MDN Web Docs, Express Tutorial Part 7: Deploying to production, Available (accessed Mar 9, 2019): https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/deployment.

[19] M.J. Kavis. Architecting the Cloud : Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS), John Wiley & Sons, Incorporated, Somerset, 2014, 224 p.

[20] National Institute of Standards and Technology, The NIST Definition of Cloud Computing, Available (accessed Mar 21, 2019): https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[21] RightScale, State of the Cloud Report, 2018, Available (accessed March 21, 2019): https://www.suse.com/media/report/rightscale_2018_state_of_the_cloud_report.pdf.

[22] Amazon Web Services, Inc., Amazon EC2, Available (accessed Mar 21, 2019): https://aws.amazon.com/ec2/.

[23] G. Rossi, Olsina L, Pastor O, Schwabe D, Web Engineering: Modelling and Implementing Web Applications, 1. Aufl. ed. Springer Verlag London Limited, 2008, 461 p.

[24] NGINX, Building a Web Frontend with Microservices and NGINX Plus, Available (accessed Jan 9, 2019): https://www.nginx.com/blog/building-a-web-frontend-with-microservices-and-nginx-plus.

[25] A. Coleman, The Inner-Workings of an MVC Web Application, Like Playing With Legos, 2014, Available (accessed Mar 14, 2019): https://selftaughtcoders.com/model-view-controller-mvc-web-application/.

[26] Mouth Media Blog, MPA or SPA? Which one should you choose? Available (accessed Mar 15, 2019): https://mouthmedia.com/blog/mpa-or-spa-which-one-should-you-choose/.

[27] React Router, Declarative Routing for React, Available (accessed Mar 15, 2019): https://reacttraining.com/react-router.

[28] A. Guryanov, What's the difference between single-page application and multi-page application? Available (accessed Mar 15, 2019): https://www.adcisolutions.com/knowledge/whats-difference-between-single-page-application-and-multi-page-application.

[29] RubyGarage, What's the Difference Between Single-Page and Multi-Page Apps, Available (accessed Mar 15, 2019): https://rubygarage.org/blog/single-page-app-vs-multi-page-app.

[30] M. Boyd, Single Page Applications: A Powerful Design Pattern for Modern Web Apps, Available (accessed Mar 15, 2019): http://www.eikospartners.com/blog/single-page-applications-a-powerful-design-pattern-for-modern-web-apps.

[31] S. Shimanovsky, Multi page web applications vs. single page web applications, Available (accessed Mar 15, 2019): http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications.

[32] J. Schatz, Our big Frontend plan revealed, 2017, Available (accessed Feb 14, 2019): https://about.gitlab.com/2017/02/06/vue-big-plan/.

[33] Node.js, Node.js, Available (accessed Feb 16, 2019): https://nodejs.org/en/.

[34] Node.js, About Node.js, Available (accessed Feb 16, 2019): https://nodejs.org/en/about.

[35] Express.js, Express - Node.js web application framework, Available (accessed Feb 25, 2019): https://expressjs.com.

[36] Redis, Redis, Available (accessed Mar 15, 2019): https://redis.io/.

[37] Redis Labs, Cache vs. Session Store, 2017, Available (accessed Mar 15, 2019): https://redislabs.com/blog/cache-vs-session-store/.

[38] W3schools, Bootstrap 4 Get Started, Available (accessed Jan 9, 2019): https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp.

[39] jQuery, What is jQuery? Available (accessed Feb 16, 2019): https://jquery.com/.

[40] React, React – A JavaScript library for building user interfaces, Available (accessed Mar 21, 2019): https://reactjs.org/index.html.

[41] J. Potter, npm trends, Available (accessed Jan 9, 2019): https://www.npmtrends.com/@angular/core-vs-angular-vs-react-vs-vue.

[42] IBM Knowledge Center, Dependencies between resources, 2014, Available (accessed Mar 22, 2019): https://www.ibm.com/support/knowledgecenter/SSPLFC_7.2.2/com.ibm.taddm.doc_7.2.2/SDK-DevGuide/c_cmdbsdk_modelobject_dependencies.html.

[43] Node.js, Dependencies, Available (accessed Mar 22, 2019): https://nodejs.org/en/docs/meta/topics/dependencies/.

[44] MDN Web Docs, Setting up a Node development environment, Available (accessed Mar 22, 2019): https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/development_environment.

[45] E.N. McKay. UI is Communication: How to Design Intuitive, User-Centered Interfaces by Focusing on Effective Communication, Elsevier Science and Technology Books, Inc, 2013, 379 p.

[46] The Code4Lib Journal, Building a Scalable and Flexible Library Data Dashboard, 2017, Available (accessed Mar 6, 2019): http://journal.code4lib.org/articles/12152.

[47] Atlassian - Jira Software Support, Configuring dashboards, Available (accessed Mar 6, 2019): https://confluence.atlassian.com/jirasoftwarecloud/configuring-dashboards-764478209.html.

[48] OWASP Github, Access Control Cheat Sheet, Available (accessed Mar 19, 2019): https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Access_Control_Cheat_Sheet.md.

[49] OWASP, Web App Access Control Design, Available (accessed Mar 18, 2019): https://www.owasp.org/images/4/41/ASDC12-Access_Control_Designs_and_Pitfalls.pdf.

[50] OWASP, OWASP Top 10 Application Security Risks - 2017, Available (accessed Feb 16, 2019): https://www.owasp.org/index.php/Top_10-2017_Top_10.

[51] Common Weakness Enumeration, CWE - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (3.2), Available (accessed March 29, 2019): https://cwe.mitre.org/data/definitions/89.html.

[52] OWASP, Top 10-2017 A4-XML External Entities (XXE), Available (accessed Mar 29, 2019): https://www.owasp.org/index.php/Top_10-2017_A4-XML_External_Entities_(XXE).

[53] Express.js, Security Best Practices for Express in Production, Available (accessed Mar 29, 2019): https://expressjs.com/en/advanced/best-practice-security.html.

[54] OWASP, Top 10-2017 A7-Cross-Site Scripting (XSS), Available (accessed Mar 31, 2019): https://www.owasp.org/index.php/Top_10-2017_A7-Cross-Site_Scripting_(XSS).

[55] OWASP, Top 10-2017 A8-Insecure Deserialization, Available (accessed Mar 31, 2019): https://www.owasp.org/index.php/Top_10-2017_A8-Insecure_Deserialization.

[56] npm, About audit reports, Available (accessed Jan 30, 2019): https://docs.npmjs.com/about-audit-reports.

[57] S. Amir-Mohammadian, Stephen Chong and Christian Skalka, Correct Audit Logging: Theory and Practice, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 139-162.

[58] OKD, What Are Deployment Strategies? Available (accessed Mar 31, 2019): https://docs.okd.io/latest/dev_guide/deployments/deployment_strategies.html#strategies.

[59] Express.js, Performance Best Practices Using Express in Production, Available (accessed Mar 31, 2019): https://expressjs.com/en/advanced/best-practice-performance.html.

[60] Github, strong-pm - Process Manager, Available (accessed April 10, 2019): https://github.com/strongloop/strong-pm.

[61] PM2, PM2 Documentation | Overview, Available (accessed April 10, 2019): https://pm2.io/doc/en/runtime/overview/.

[62] M. Saunders, Lewis P, Thornhill A, Research methods for business students, 5th ed. Prentice Hall, Harlow, 2009, 614 p.

[63] M. J. Aamir and A. Mansoor, Testing Web Application from usability perspective, 2013 3rd IEEE International Conference on Computer, Control and Communication (IC4), pp. 1-7.